4. COMPOSER_BIN_DIR
5. http_proxy or HTTP_PROXY
6. no_proxy
7. HTTP_PROXY_REQUEST_FULLURI
8. HTTPS_PROXY_REQUEST_FULLURI
9. COMPOSER_HOME
    1. COMPOSER_HOME/config.json
10. COMPOSER_CACHE_DIR
11. COMPOSER_PROCESS_TIMEOUT
12. COMPOSER_CAFILE
13. COMPOSER_AUTH
14. COMPOSER_DISCARD_CHANGES
15. COMPOSER_NO_INTERACTION
16. COMPOSER_DISABLE_XDEBUG_WARN
17. COMPOSER_ALLOW_SUPERUSER
18. COMPOSER_MIRROR_PATH_REPOS

Command-line interface / Commands

You've already learned how to use the command-line interface to do some things. This chapter documents all the available commands.

To get help from the command-line, simply call `composer` or `composer list` to see the complete list of commands, then `--help` combined with any of those can give you more information.

Global Options

The following options are available with every command:

- **--verbose (-v):** Increase verbosity of messages.
- **--help (-h):** Display help information.
- **--quiet (-q):** Do not output any message.
- **--no-interaction (-n):** Do not ask any interactive question.
- **--no-plugins:** Disables plugins.
- **--working-dir (-d):** If specified, use the given directory as working directory.
- **--profile:** Display timing and memory usage information
- **--ansi:** Force ANSI output.
- **--no-ansi:** Disable ANSI output.
- **--version (-V):** Display this application version.

Process Exit Codes

- **0:** OK

- **1:** Generic/unknown error code
- **2:** Dependency solving error code

init

In the Libraries (02-libraries.md) chapter we looked at how to create a `composer.json` by hand. There is also an `init` command available that makes it a bit easier to do this.

When you run the command it will interactively ask you to fill in the fields, while using some smart defaults.

```
php composer.phar init
```

Options

- **--name:** Name of the package.
- **--description:** Description of the package.
- **--author:** Author name of the package.
- **--homepage:** Homepage of the package.
- **--require:** Package to require with a version constraint. Should be in format `foo/bar:1.0.0`.
- **--require-dev:** Development requirements, see **--require**.
- **--stability (-s):** Value for the `minimum-stability` field.
- **--repository:** Provide one (or more) custom repositories. They will be stored in the generated composer.json, and used for auto-completion when prompting for the list of requires. Every repository can be either an HTTP URL pointing to a `composer` repository or a JSON string which similar to what the repositories (04-schema.md#repositories) key accepts.

install

The `install` command reads the `composer.json` file from the current directory, resolves the dependencies, and installs them into `vendor`.

```
php composer.phar install
```

If there is a `composer.lock` file in the current directory, it will use the exact versions from there instead of resolving them. This ensures that everyone using the library will get the same versions of the dependencies.

If there is no `composer.lock` file, Composer will create one after dependency resolution.

Options

- **--prefer-source:** There are two ways of downloading a package: `source` and `dist`. For stable versions Composer will use the `dist` by default. The `source` is a version control repository. If `--prefer-source` is enabled, Composer will install from `source` if there is one. This is useful if you want to make a bugfix to a project and get a local git clone of the dependency directly.
- **--prefer-dist:** Reverse of `--prefer-source`, Composer will install from `dist` if possible. This can speed up installs substantially on build servers and other use cases where you typically do not run updates of the vendors. It is also a way to circumvent problems with git if you do not have a proper setup.
- **--ignore-platform-reqs:** ignore `php`, `hhvm`, `lib-*` and `ext-*` requirements and force the installation even if the local machine does not fulfill these. See also the `platform` (06-config.md#platform) config option.
- **--dry-run:** If you want to run through an installation without actually installing a package, you can use `--dry-run`. This will simulate the installation and show you what would happen.
- **--dev:** Install packages listed in `require-dev` (this is the default behavior).
- **--no-dev:** Skip installing packages listed in `require-dev`. The autoloader generation skips the `autoload-dev` rules.
- **--no-autoloader:** Skips autoloader generation.
- **--no-scripts:** Skips execution of scripts defined in `composer.json`.
- **--no-progress:** Removes the progress display that can mess with some terminals or scripts which don't handle backspace characters.
- **--no-suggest:** Skips suggested packages in the output.
- **--optimize-autoloader (-o):** Convert PSR-0/4 autoloading to classmap to get a faster autoloader. This is recommended especially for production, but can take a bit of time to run so it is currently not done by default.
- **--classmap-authoritative (-a):** Autoload classes from the classmap only. Implicitly enables `--optimize-autoloader`.

update

In order to get the latest versions of the dependencies and to update the `composer.lock` file, you should use the `update` command.

```
php composer.phar update
```

This will resolve all dependencies of the project and write the exact versions into `composer.lock`.

If you just want to update a few packages and not all, you can list them as such:

```
php composer.phar update vendor/package vendor/package2
```

You can also use wildcards to update a bunch of packages at once:

```
php composer.phar update vendor/*
```

Options

- **--prefer-source:** Install packages from `source` when available.
- **--prefer-dist:** Install packages from `dist` when available.
- **--ignore-platform-reqs:** ignore `php`, `hhvm`, `lib-*` and `ext-*` requirements and force the installation even if the local machine does not fulfill these. See also the `platform` (06-config.md#platform) config option.
- **--dry-run:** Simulate the command without actually doing anything.
- **--dev:** Install packages listed in `require-dev` (this is the default behavior).
- **--no-dev:** Skip installing packages listed in `require-dev`. The autoloader generation skips the `autoload-dev` rules.
- **--no-autoloader:** Skips autoloader generation.
- **--no-scripts:** Skips execution of scripts defined in `composer.json`.
- **--no-progress:** Removes the progress display that can mess with some terminals or scripts which don't handle backspace characters.
- **--no-suggest:** Skips suggested packages in the output.
- **--optimize-autoloader (-o):** Convert PSR-0/4 autoloading to classmap to get a faster autoloader. This is recommended especially for production, but can take a bit of time to run so it is currently not done by default.
- **--classmap-authoritative (-a):** Autoload classes from the classmap only. Implicitly enables `--optimize-autoloader`.
- **--lock:** Only updates the lock file hash to suppress warning about the lock file being out of date.
- **--with-dependencies:** Add also all dependencies of whitelisted packages to the whitelist.
- **--root-reqs:** Restricts the update to your first degree dependencies.
- **--prefer-stable:** Prefer stable versions of dependencies.
- **--prefer-lowest:** Prefer lowest versions of dependencies. Useful for testing minimal versions of requirements, generally used with `--prefer-stable`.

require

The `require` command adds new packages to the `composer.json` file from the current directory. If no file exists one will be created on the fly.

```
php composer.phar require
```

After adding/changing the requirements, the modified requirements will be installed or updated.

If you do not want to choose requirements interactively, you can just pass them to the command.

```
php composer.phar require vendor/package:2.* vendor/package2:dev-master
```

Options

- **--prefer-source:** Install packages from `source` when available.
- **--prefer-dist:** Install packages from `dist` when available.
- **--ignore-platform-reqs:** ignore `php`, `hhvm`, `lib-*` and `ext-*` requirements and force the installation even if the local machine does not fulfill these. See also the `platform` (06-config.md#platform) config option.
- **--dev:** Add packages to `require-dev`.
- **--no-update:** Disables the automatic update of the dependencies.
- **--no-progress:** Removes the progress display that can mess with some terminals or scripts which don't handle backspace characters.
- **--no-suggest:** Skips suggested packages in the output.
- **--no-scripts:** Skips execution of scripts defined in `composer.json`.
- **--update-no-dev:** Run the dependency update with the `--no-dev` option.
- **--update-with-dependencies:** Also update dependencies of the newly required packages.
- **--sort-packages:** Keep packages sorted in `composer.json`.
- **--optimize-autoloader (-o):** Convert PSR-0/4 autoloading to classmap to get a faster autoloader. This is recommended especially for production, but can take a bit of time to run so it is currently not done by default.
- **--classmap-authoritative (-a):** Autoload classes from the classmap only. Implicitly enables `--optimize-autoloader`.
- **--prefer-stable:** Prefer stable versions of dependencies.
- **--prefer-lowest:** Prefer lowest versions of dependencies. Useful for testing minimal versions of requirements, generally used with `--prefer-stable`.

remove

The `remove` command removes packages from the `composer.json` file from the current directory.

```
php composer.phar remove vendor/package vendor/package2
```

After removing the requirements, the modified requirements will be uninstalled.

Options

- **--ignore-platform-reqs:** ignore `php`, `hhvm`, `lib-*` and `ext-*` requirements and force the installation even if the local machine does not fulfill these. See also the `platform` (06-config.md#platform) config option.
- **--dev:** Remove packages from `require-dev`.

- **--no-update:** Disables the automatic update of the dependencies.
- **--no-progress:** Removes the progress display that can mess with some terminals or scripts which don't handle backspace characters.
- **--no-scripts:** Skips execution of scripts defined in `composer.json`.
- **--update-no-dev:** Run the dependency update with the --no-dev option.
- **--update-with-dependencies:** Also update dependencies of the removed packages.
- **--optimize-autoloader (-o):** Convert PSR-0/4 autoloading to classmap to get a faster autoloader. This is recommended especially for production, but can take a bit of time to run so it is currently not done by default.
- **--classmap-authoritative (-a):** Autoload classes from the classmap only. Implicitly enables `--optimize-autoloader`.

global

The global command allows you to run other commands like `install`, `require` or `update` as if you were running them from the COMPOSER_HOME directory.

This is merely a helper to manage a project stored in a central location that can hold CLI tools or Composer plugins that you want to have available everywhere.

This can be used to install CLI utilities globally. Here is an example:

```
php composer.phar global require fabpot/php-cs-fixer
```

Now the `php-cs-fixer` binary is available globally. Just make sure your global vendor binaries (articles/vendor-binaries.md) directory is in your `$PATH` environment variable, you can get its location with the following command :

```
php composer.phar global config bin-dir --absolute
```

If you wish to update the binary later on you can just run a global update:

```
php composer.phar global update
```

search

The search command allows you to search through the current project's package repositories. Usually this will be just packagist. You simply pass it the terms you want to search for.

```
php composer.phar search monolog
```

You can also search for more than one term by passing multiple arguments.

Options

- **--only-name (-N):** Search only in name.

show

To list all of the available packages, you can use the `show` command.

```
php composer.phar show
```

To filter the list you can pass a package mask using wildcards.

```
php composer.phar show monolog/*

monolog/monolog 1.19.0 Sends your logs to files, sockets, inboxes, databases and va
```

If you want to see the details of a certain package, you can pass the package name.

```
php composer.phar show monolog/monolog

name     : monolog/monolog
versions : master-dev, 1.0.2, 1.0.1, 1.0.0, 1.0.0-RC1
type     : library
names    : monolog/monolog
source   : [git] https://github.com/Seldaek/monolog.git 3d4e60d0cbc4b888fe5ad223d77
dist     : [zip] https://github.com/Seldaek/monolog/zipball/3d4e60d0cbc4b888fe5ad22
license  : MIT

autoload
psr-0
Monolog : src/

requires
php >=5.3.0
```

You can even pass the package version, which will tell you the details of that specific version.

```
php composer.phar show monolog/monolog 1.0.2
```

Options

- **--latest (-l):** List all installed packages including their latest version.
- **--all (-a):** List all packages available in all your repositories.
- **--installed (-i):** List the packages that are installed (this is enabled by default, and deprecated).
- **--platform (-p):** List only platform packages (php & extensions).
- **--self (-s):** List the root package info.
- **--tree (-t):** List your dependencies as a tree. If you pass a package name it will show the dependency tree for that package.
- **--name-only (-N):** List package names only.
- **--path (-P):** List package paths.
- **--outdated (-o):** Implies --latest, but this lists *only* packages that have a newer version available.
- **--minor-only (-m):** Use with --latest. Only shows packages that have minor SemVer-compatible updates.
- **--direct (-D):** Restricts the list of packages to your direct dependencies.

outdated

The `outdated` command shows a list of installed packages that have updates available, including their current and latest versions. This is basically an alias for `composer show -lo`.

The color coding is as such:

- **green**: Dependency is in the latest version and is up to date.
- **yellow**: Dependency has a new version available that includes backwards compatibility breaks according to semver, so upgrade when you can but it may involve work.
- **red**: Dependency has a new version that is semver-compatible and you should upgrade it.

Options

- **--all (-a):** Show all packages, not just outdated (alias for `composer show -l`).
- **--direct (-D):** Restricts the list of packages to your direct dependencies.
- **--minor-only (-m):** Only shows packages that have minor SemVer-compatible updates.

browse / home

The `browse` (aliased to `home`) opens a package's repository URL or homepage in your browser.

Options

- **--homepage (-H):** Open the homepage instead of the repository URL.

suggests

Lists all packages suggested by currently installed set of packages. You can optionally pass one or multiple package names in the format of `vendor/package` to limit output to suggestions made by those packages only.

Use the `--by-package` or `--by-suggestion` flags to group the output by the package offering the suggestions or the suggested packages respectively.

Options

- **--by-package:** Groups output by suggesting package.
- **--by-suggestion:** Groups output by suggested package.
- **--no-dev:** Excludes suggestions from `require-dev` packages.
- **--verbose (-v):** Increased verbosity adds suggesting package name and reason for suggestion.

depends

The `depends` command tells you which other packages depend on a certain package. As with installation `require-dev` relationships are only considered for the root package.

```
php composer.phar depends doctrine/lexer
 doctrine/annotations v1.2.7 requires doctrine/lexer (1.*)
 doctrine/common      v2.6.1 requires doctrine/lexer (1.*)
```

You can optionally specify a version constraint after the package to limit the search.

Add the `--tree` or `-t` flag to show a recursive tree of why the package is depended upon, for example:

```
php composer.phar depends psr/log -t
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)
```

Options

- **--recursive (-r):** Recursively resolves up to the root package.
- **--tree (-t):** Prints the results as a nested tree, implies -r.

prohibits

The `prohibits` command tells you which packages are blocking a given package from being installed. Specify a version constraint to verify whether upgrades can be performed in your project, and if not why not. See the following example:

```
php composer.phar prohibits symfony/symfony 3.1
 laravel/framework v5.2.16 requires symfony/var-dumper (2.8.*|3.0.*)
```

Note that you can also specify platform requirements, for example to check whether you can upgrade your server to PHP 8.0:

```
php composer.phar prohibits php:8
 doctrine/cache        v1.6.0 requires php (~5.5|~7.0)
 doctrine/common       v2.6.1 requires php (~5.5|~7.0)
 doctrine/instantiator 1.0.5  requires php (>=5.3,<8.0-DEV)
```

As with `depends` you can request a recursive lookup, which will list all packages depending on the packages that cause the conflict.

Options

- **--recursive (-r):** Recursively resolves up to the root package.
- **--tree (-t):** Prints the results as a nested tree, implies -r.

validate

You should always run the `validate` command before you commit your `composer.json` file, and before you tag a release. It will check if your `composer.json` is valid.

```
php composer.phar validate
```

Options

- **--no-check-all:** Do not emit a warning if requirements in `composer.json` use unbound version constraints.
- **--no-check-lock:** Do not emit an error if `composer.lock` exists and is not up to date.

- **--no-check-publish:** Do not emit an error if `composer.json` is unsuitable for publishing as a package on Packagist but is otherwise valid.

status

If you often need to modify the code of your dependencies and they are installed from source, the `status` command allows you to check if you have local changes in any of them.

```
php composer.phar status
```

With the `--verbose` option you get some more information about what was changed:

```
php composer.phar status -v

You have changes in the following dependencies:
vendor/seld/jsonlint:
    M README.mdown
```

self-update

To update Composer itself to the latest version, just run the `self-update` command. It will replace your `composer.phar` with the latest version.

```
php composer.phar self-update
```

If you would like to instead update to a specific release simply specify it:

```
php composer.phar self-update 1.0.0-alpha7
```

If you have installed Composer for your entire system (see global installation (00-intro.md#globally)), you may have to run the command with `root` privileges

```
sudo -H composer self-update
```

Options

- **--rollback (-r):** Rollback to the last version you had installed.
- **--clean-backups:** Delete old backups during an update. This makes the current version of Composer the only backup available after the update.

config

The `config` command allows you to edit composer config settings and repositories in either the local `composer.json` file or the global `config.json` file.

Additionally it lets you edit most properties in the local `composer.json`.

```
php composer.phar config --list
```

Usage

`config [options] [setting-key] [setting-value1] ... [setting-valueN]`

`setting-key` is a configuration option name and `setting-value1` is a configuration value. For settings that can take an array of values (like `github-protocols`), more than one setting-value arguments are allowed.

You can also edit the values of the following properties:

`description`, `homepage`, `keywords`, `license`, `minimum-stability`, `name`, `prefer-stable`, `type` and `version`.

See the Config (06-config.md) chapter for valid configuration options.

Options

- **--global (-g):** Operate on the global config file located at `$COMPOSER_HOME/config.json` by default. Without this option, this command affects the local composer.json file or a file specified by `--file`.
- **--editor (-e):** Open the local composer.json file using in a text editor as defined by the `EDITOR` env variable. With the `--global` option, this opens the global config file.
- **--unset:** Remove the configuration element named by `setting-key`.
- **--list (-l):** Show the list of current config variables. With the `--global` option this lists the global configuration only.
- **--file="..." (-f):** Operate on a specific file instead of composer.json. Note that this cannot be used in conjunction with the `--global` option.
- **--absolute:** Returns absolute paths when fetching *-dir config values instead of relative.

Modifying Repositories

In addition to modifying the config section, the `config` command also supports making changes to the repositories section by using it the following way:

```
php composer.phar config repositories.foo vcs https://github.com/foo/bar
```

If your repository requires more configuration options, you can instead pass its JSON representation :

```
php composer.phar config repositories.foo '{"type": "vcs", "url": "http://svn.examp
```

Modifying Extra Values

In addition to modifying the config section, the `config` command also supports making changes to the extra section by using it the following way:

```
php composer.phar config extra.foo.bar value
```

The dots indicate array nesting, a max depth of 3 levels is allowed though. The above would set `"extra": { "foo": { "bar": "value" } }`.

create-project

You can use Composer to create new projects from an existing package. This is the equivalent of doing a git clone/svn checkout followed by a "composer install" of the vendors.

There are several applications for this:

1. You can deploy application packages.
2. You can check out any package and start developing on patches for example.
3. Projects with multiple developers can use this feature to bootstrap the initial application for development.

To create a new project using Composer you can use the "create-project" command. Pass it a package name, and the directory to create the project in. You can also provide a version as third argument, otherwise the latest version is used.

If the directory does not currently exist, it will be created during installation.

```
php composer.phar create-project doctrine/orm path 2.2.*
```

It is also possible to run the command without params in a directory with an existing `composer.json` file to bootstrap a project.

By default the command checks for the packages on packagist.org.

Options

- **--repository:** Provide a custom repository to search for the package, which will be used instead of packagist. Can be either an HTTP URL pointing to a `composer` repository, a path to a local `packages.json` file, or a JSON string which similar to what the repositories (04-schema.md#repositories) key accepts.
- **--stability (-s):** Minimum stability of package. Defaults to `stable`.
- **--prefer-source:** Install packages from `source` when available.
- **--prefer-dist:** Install packages from `dist` when available.
- **--dev:** Install packages listed in `require-dev`.
- **--no-install:** Disables installation of the vendors.
- **--no-scripts:** Disables the execution of the scripts defined in the root package.
- **--no-progress:** Removes the progress display that can mess with some terminals or scripts which don't handle backspace characters.
- **--keep-vcs:** Skip the deletion of the VCS metadata for the created project. This is mostly useful if you run the command in non-interactive mode.
- **--ignore-platform-reqs:** ignore `php`, `hhvm`, `lib-*` and `ext-*` requirements and force the installation even if the local machine does not fulfill these.

dump-autoload

If you need to update the autoloader because of new classes in a classmap package for example, you can use "dump-autoload" to do that without having to go through an install or update.

Additionally, it can dump an optimized autoloader that converts PSR-0/4 packages into classmap ones for performance reasons. In large applications with many classes, the autoloader can take up a substantial portion of every request's time. Using classmaps for everything is less convenient in development, but using this option you can still use PSR-0/4 for convenience and classmaps for performance.

Options

- **--optimize (-o):** Convert PSR-0/4 autoloading to classmap to get a faster autoloader. This is recommended especially for production, but can take a bit of time to run so it is currently not done by default.
- **--classmap-authoritative (-a):** Autoload classes from the classmap only. Implicitly enables `--optimize`.
- **--no-dev:** Disables autoload-dev rules.

clear-cache

Deletes all content from Composer's cache directories.

licenses

Lists the name, version and license of every package installed. Use `--format=json` to get machine readable output.

Options

- **--no-dev:** Remove dev dependencies from the output
- **--format:** Format of the output: text or json (default: "text")

run-script

Options

- **--timeout:** Set the script timeout in seconds, or 0 for no timeout.
- **--no-dev:** Disable dev mode
- **--list:** List user defined scripts

To run scripts (articles/scripts.md) manually you can use this command, just give it the script name and optionally any required arguments.

exec

Executes a vendored binary/script. You can execute any command and this will ensure that the Composer bin-dir is pushed on your PATH before the command runs.

Options

- **--list:** List the available composer binaries

diagnose

If you think you found a bug, or something is behaving strangely, you might want to run the `diagnose` command to perform automated checks for many common problems.

```
php composer.phar diagnose
```

archive

This command is used to generate a zip/tar archive for a given package in a given version. It can also be used to archive your entire project without excluded/ignored files.

```
php composer.phar archive vendor/package 2.0.21 --format=zip
```

Options

- **--format (-f):** Format of the resulting archive: tar or zip (default: "tar")
- **--dir:** Write the archive to this directory (default: ".")

help

To get more information about a certain command, just use `help`.

```
php composer.phar help install
```

Command-line completion

Command-line completion can be enabled by following instructions on this page (https://github.com/bamarni/symfony-console-autocomplete).

Environment variables

You can set a number of environment variables that override certain settings. Whenever possible it is recommended to specify these settings in the `config` section of `composer.json` instead. It is worth noting that the env vars will always take precedence over the values specified in `composer.json`.

COMPOSER

By setting the `COMPOSER` env variable it is possible to set the filename of `composer.json` to something else.

For example:

```
COMPOSER=composer-other.json php composer.phar install
```

The generated lock file will use the same name: `composer-other.lock` in this example.

COMPOSER_ROOT_VERSION

By setting this var you can specify the version of the root package, if it can not be guessed from VCS info and is not present in `composer.json`.

COMPOSER_VENDOR_DIR

By setting this var you can make Composer install the dependencies into a directory other than `vendor`.

COMPOSER_BIN_DIR

By setting this option you can change the `bin` (Vendor Binaries (articles/vendor-binaries.md)) directory to something other than `vendor/bin`.

http_proxy or HTTP_PROXY

If you are using Composer from behind an HTTP proxy, you can use the standard `http_proxy` or `HTTP_PROXY` env vars. Simply set it to the URL of your proxy. Many operating systems already set this variable for you.

Using `http_proxy` (lowercased) or even defining both might be preferable since some tools like git or curl will only use the lower-cased `http_proxy` version. Alternatively you can also define the git proxy using `git config --global http.proxy <proxy url>`.

If you are using Composer in a non-CLI context (i.e. integration into a CMS or similar use case), and need to support proxies, please provide the `CGI_HTTP_PROXY` environment variable instead. See httpoxy.org (https://httpoxy.org/) for further details.

no_proxy

If you are behind a proxy and would like to disable it for certain domains, you can use the `no_proxy` env var. Simply set it to a comma separated list of domains the proxy should *not* be used for.

The env var accepts domains, IP addresses, and IP address blocks in CIDR notation. You can restrict the filter to a particular port (e.g. `:80`). You can also set it to `*` to ignore the proxy for all HTTP requests.

HTTP_PROXY_REQUEST_FULLURI

If you use a proxy but it does not support the request_fulluri flag, then you should set this env var to `false` or `0` to prevent Composer from setting the request_fulluri option.

HTTPS_PROXY_REQUEST_FULLURI

If you use a proxy but it does not support the request_fulluri flag for HTTPS requests, then you should set this env var to `false` or `0` to prevent Composer from setting the request_fulluri option.

COMPOSER_HOME

The `COMPOSER_HOME` var allows you to change the Composer home directory. This is a hidden, global (per-user on the machine) directory that is shared between all projects.

By default it points to `C:\Users\<user>\AppData\Roaming\Composer` on Windows and `/Users/<user>/.composer` on OSX. On *nix systems that follow the XDG Base Directory Specifications (http://standards.freedesktop.org/basedir-spec/basedir-spec-latest.html), it points to `$XDG_CONFIG_HOME/composer`. On other* nix systems, it points to `/home/<user>/.composer`.

COMPOSER_HOME/config.json

You may put a `config.json` file into the location which `COMPOSER_HOME` points to. Composer will merge this configuration with your project's `composer.json` when you run the `install` and `update` commands.

This file allows you to set repositories (05-repositories.md) and configuration (06-config.md) for the user's projects.

In case global configuration matches *local* configuration, the *local* configuration in the project's `composer.json` always wins.

COMPOSER_CACHE_DIR

The `COMPOSER_CACHE_DIR` var allows you to change the Composer cache directory, which is also configurable via the `cache-dir` (06-config.md#cache-dir) option.

By default it points to $COMPOSER_HOME/cache on *nix and OSX, and `C:\Users\<user>\AppData\Local\Composer` (or `%LOCALAPPDATA%/Composer`) on Windows.

COMPOSER_PROCESS_TIMEOUT

This env var controls the time Composer waits for commands (such as git commands) to finish executing. The default value is 300 seconds (5 minutes).

COMPOSER_CAFILE

By setting this environmental value, you can set a path to a certificate bundle file to be used during SSL/TLS peer verification.

COMPOSER_AUTH

The `COMPOSER_AUTH` var allows you to set up authentication as an environment variable. The contents of the variable should be a JSON formatted object containing http-basic, github-oauth, bitbucket-oauth, ... objects as needed, and following the spec from the config (06-config.md#gitlab-oauth).

COMPOSER_DISCARD_CHANGES

This env var controls the `discard-changes` (06-config.md#discard-changes) config option.

COMPOSER_NO_INTERACTION

If set to 1, this env var will make Composer behave as if you passed the `--no-interaction` flag to every command. This can be set on build boxes/CI.

COMPOSER_DISABLE_XDEBUG_WARN

If set to 1, this env disables the warning about having xdebug enabled.

COMPOSER_ALLOW_SUPERUSER

If set to 1, this env disables the warning about running commands as root/super user. It also disables automatic clearing of sudo sessions, so you should really only set this if you use Composer as super user at all times like in docker containers.

COMPOSER_MIRROR_PATH_REPOS

If set to 1, this env changes the default path repository strategy to `mirror` instead of `symlink`. As it is the default strategy being set it can still be overwritten by repository options.

← Libraries (02-libraries.md) | Schema (04-schema.md) →

Found a typo? Something is wrong in this documentation? Just fork and edit (http://github.com/composer/composer/edit/master/doc/03-cli.md) it!