Composer Cheatsheet

A comprehensive cheat sheet for Composer, the dependency manager for PHP. Covers installation, dependency management, autoloading, and common commands with examples.

Installation & Basic Usage

Installation

Local Installation:

Download `composer.phar` and place it in your project directory.

Global Installation:

```
curl -sS
https://getcomposer.org/installer | php
mv composer.phar /usr/local/bin/composer
chmod +x /usr/local/bin/composer
```

Verify Installation:

```
composer --version
```

Ensure that PHP is installed and accessible from your command line.

Basic Commands

`composer init`	Initializes a new composer project in the current directory.
`composer install`	Installs the project's dependencies from the `composer.lock` file, or `composer.json` if `composer.lock` doesn't exist.
`composer update`	Updates the project's dependencies to the latest versions specified in `composer.json` and updates the `composer.lock` file.
`composer require <package>`	Adds a new dependency to the `composer.json` file and installs it.
`composer remove <package>`	Removes a dependency from the `composer.json` file and uninstalls it.
`composer dump-autoload`	Regenerates the autoloader files.

composer.json Structure

```
{
  "name": "vendor/package-name",
  "description": "A short description of the package.",
  "type": "project",
  "license": "MIT",
  "require": {
    "php": ">=7.4",
    "vendor/dependency": "^1.0"
  },
  "autoload": {
    "psr-4": {
      "Namespace\\": "src/"
    }
  },
  "require-dev": {
    "phpunit/phpunit": "^9.0"
  },
  "scripts": {
    "test": "phpunit"
  }
}
```

Dependency Management

Specifying Versions

`1.2.3`	Specific version.
`>=1.2.3`	Minimum version, allows later versions.
`<1.2.3`	Maximum version, allows earlier versions.
`~1.2.3`	Equivalent to `>=1.2.3 <1.3.0`. Allows updates up to the next minor version.
`^1.2.3`	Equivalent to `>=1.2.3 <2.0.0`. Allows updates until the next major version.
`*`	Any version. Not recommended for production.
`dev-master`	Install the latest code from the `master` branch. Unstable.

Updating Dependencies

`composer update` updates your dependencies to the latest versions according to the constraints specified in your `composer.json` file.

Always commit your `composer.lock` file after updating dependencies.

If you want to update only single package use `composer update vendor/package`

Resolving Conflicts

If Composer encounters conflicts, it will provide error messages suggesting how to resolve them. This often involves relaxing version constraints in your `composer.json` file.

Use `composer diagnose` to identify common configuration issues.

Consider using the `composer prohibits <package> <version>` command to understand why a package cannot be installed.

Autoloading

PSR-4 Autoloading

PSR-4 is the recommended autoloading standard.

Specify the namespace to directory mapping in the `autoload` section of `composer.json`:

```
"autoload": {
  "psr-4": {
    "MyProject\\": "src/"
  }
}
```

This maps the `MyProject` namespace to the `src` directory.

After modifying the `autoload` section, run `composer dump-autoload` to regenerate the autoloader.

Classmap Autoloading

Classmap autoloading scans specified directories for PHP classes and builds a map.

```
"autoload": {
  "classmap": [
    "src/",
    "lib/"
  ]
}
```

Run `composer dump-autoload` after modifying the `classmap` section.

Files Autoloading

Files autoloading includes specified PHP files.

```
"autoload": {
  "files": [
    "src/helpers.php",
    "lib/config.php"
  ]
}
```

Run `composer dump-autoload` after modifying the `files` section.

Optimizing Autoloading

Use the `-o` or `--optimize` option with `composer dump-autoload` to generate an optimized autoloader for production:

```
composer dump-autoload -o
```

This generates a single `autoload_static.php` file which can improve performance.

Advanced Usage

Scripts

Define custom scripts in the `scripts` section of `composer.json`:

```
"scripts": {
  "test": "phpunit",
  "lint": "phpcs --standard=PSR12 src/"
}
```

Run scripts using `composer <script-name>`:

```
composer test
composer lint
```

Repositories

Configure custom repositories in the `repositories` section of `composer.json` to include packages from alternative sources:

```
"repositories": [
  {
    "type": "vcs",
    "url": "https://github.com/my-org/private-repo"
  }
]
```

Supported repository types include `vcs` (version control system), `package`, `composer`, and `path`.

Platform Configuration

Specify platform requirements in the `config` section of `composer.json` to override the detected environment:

```
"config": {
  "platform": {
    "php": "7.4",
    "ext-intl": "7.4"
  }
}
```

This is useful for ensuring compatibility on different environments.

Plugins

Composer plugins extend Composer's functionality. Install plugins like any other dependency.

Ensure `composer/installers` is required in your project to handle different package types:

```
composer require composer/installers
```