COMPOSER

USO DE COMPOSER: Comandos y Esquema

Si la instalación fue correcta, podemos comprobarlo con el comando composer desde un terminal, tanto en Windows como en Linux. Con la opción -V muestra la versión:

> composer -V

```
alumno@alumno-VirtualBox:~$ composer -V
Composer version 2.6.6 2023-12-08 18:32:26
```

Los comandos de composer se ejecutan desde un terminal, en el directorio del proyecto, de la forma:

$ composer comando

COMANDO INIT:

La maravilla del comando init es que, a partir de un directorio vacío, genera una estructura de proyecto, crea un manifiesto con las dependencias que le pidamos, e instala todas las dependencias. Nuestro proyecto queda así iniciado con las dependencias instaladas.

Para ejecutarlo:

$ composer init

Este comando es un asistente, nos ayuda a crear el fichero de configuración **composer.json**, que define las dependencias del proyecto.

.

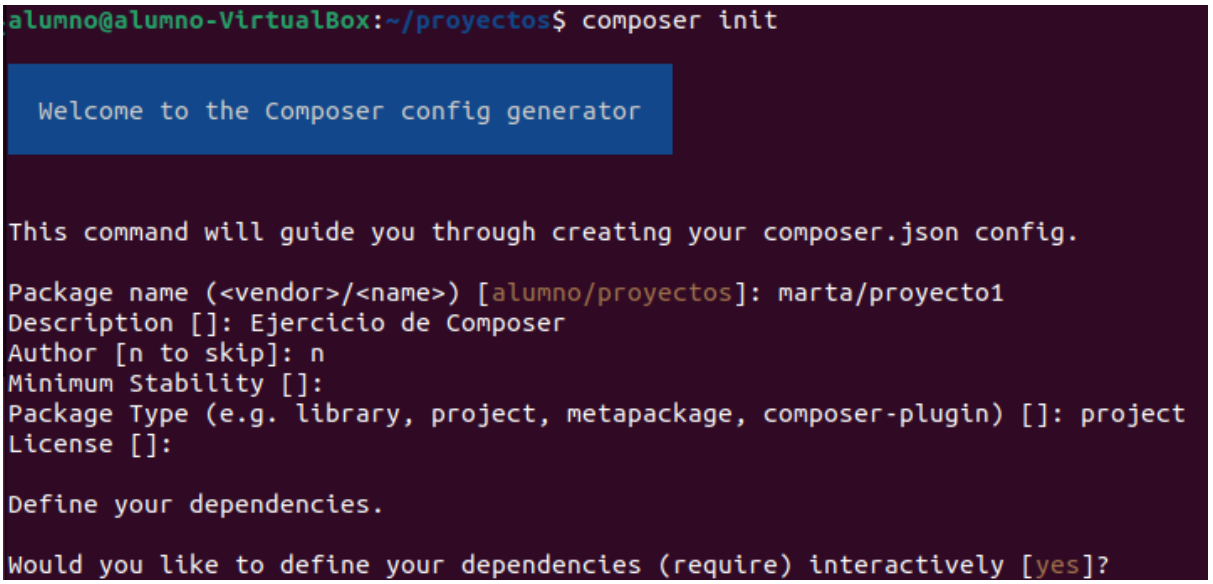
```
alumno@alumno-VirtualBox:~$ composer init

  Welcome to the Composer config generator


This command will guide you through creating your composer.json config.

Package name (<vendor>/<name>) [alumno/alumno]:
```

composer.json es el archivo manifiesto que estará en mi proyecto y contendrá todas las dependencias del mismo. Este archivo sigue un esquema definido, y mediante el comando init podemos crearlo fácilmente.

Configuración del proyecto:

Lo primero que pide es que asignemos un nombre de paquete a nuestro proyecto, y nos recomienda que utilicemos la nomenclatura <vendor>/<name> (vendedor/nombre paquete).

1. Vamos a utilizar como nombre de empresa, alumno, y como nombre de paquete, composer-workshop.
2. Luego pide una descripción, que es opcional. Pongo como descripción “Ejercicio de Composer”.
3. Luego nos pide el autor, con un formato determinado: Pepe Gómez <pepe@eu.es>. Podemos o no poner el autor, si ponemos “n” no quedará autor informado.
4. Luego nos pide la estabilidad mínima, por defecto se define como estable, así que no ponemos nada, damos al enter. Lo normal es que queramos mantener las dependencias estables, para prevenir mal funcionamiento de alguna de ellas.
5. Nos pide el top de paquete (librería, proyecto, plugin de composer, metapaquete, etc). Lo habitual son librerías y proyectos. Los metapaquetes son para librerías de composer, y los plugin son para añadir o extender la funcionalidad del propio composer. Elegimos lo más habitual, que es un proyecto: “project”.
6. Luego pide licencia, opcional, así que no la ponemos.



Añadir paquetes:
En la siguiente sección, init nos ofrece una forma fácil de añadir paquetes de librerías para usar en nuestra aplicación.

Nos pregunta si queremos definir las dependencias de forma interactiva (dentro de require, que es un apartado del manifiesto con las librerías requeridas). Por defecto tiene que sí, así que hacemos enter. Esto abrirá un asistente con el siguiente prompt:

Search for a package:

Es un buscador de paquetes. Los paquetes son definidos como **repositorio/paquete**
Un repositorio habitual para PHP es doctrine, así que ponemos doctrine, de forma que el asistente nos mostrará los paquetes más habituales que están en dicho repositorio, y también nos mostrará paquetes con la palabra doctrine dentro de otros repositorios:

```
Define your dependencies.

Would you like to define your dependencies (require) interactively [yes]?
Search for a package: doctrine

Found 15 packages matching doctrine

   [0] doctrine/orm
   [1] doctrine/lexer
   [2] doctrine/instantiator
   [3] doctrine/inflector
   [4] doctrine/event-manager
   [5] doctrine/dbal
   [6] doctrine/cache
   [7] doctrine/annotations
   [8] gedmo/doctrine-extensions
   [9] doctrine/doctrine-migrations-bundle
  [10] doctrine/doctrine-fixtures-bundle
  [11] doctrine/doctrine-bundle
  [12] symfony/doctrine-bridge
  [13] symfony/property-info
  [14] doctrine/common

Enter package # to add, or the complete package name if it is not listed:
```

Uno de los paquetes más comunes es el **dbal** (para gestión de base de datos). Escribo doctrine/dbal para que añada el paquete como dependencia de mi proyecto.
Cuando elijo un paquete, éste puede tener muchas versiones. Init me pregunta qué **versión** queremos utilizar, si lo dejo en blanco, añadirá la última versión. Luego, nos indica la versión añadida, y nos pide más paquetes.

```
Enter package # to add, or the complete package name if it is not listed: doctrine/dbal
Enter the version constraint to require (or leave blank to use the latest version):
Using version ^3.7 for doctrine/dbal
Search for a package:
```

El “sombrero” indica que a partir de la **^**3.7 de dbal. Composer utilizará las dependencias de esa versión en adelante, nunca las anteriores, es lo que se llama **restricción de versiones**.
Después nos pregunta si queremos añadir más paquetes, y lo haríamos de la misma forma. Si ya no queremos añadir más paquetes, damos al enter y finalizamos esta etapa de añadir paquetes.

<u>Dependencias de desarrollo:</u>
En ocasiones necesitamos librerías que facilitan el proceso de desarrollo, pero no son necesarias para el funcionamiento de nuestra aplicación. Un ejemplo es **phpunit**, para pruebas unitarias.
Init nos pregunta si queremos definir nuestras **dependencias del proyecto** *en desarrollo*. Estas dependencias NO SON las de nuestro proyecto, sino las que vamos a utilizar únicamente durante la fase de desarrollo. Éstas dependencias **no** se instalarán durante un despliegue en producción con Composer. Decimos que sí queremos añadir dependencias haciendo enter ya que yes es la opción por defecto.

A continuación muestra el buscador de paquetes de antes, para añadir estos paquetes nuevos como dependencias de desarrollo.
Un ejemplo de dependencia en desarrollo es **phpunit**, para pruebas unitarias con php. Se la ponemos para añadirla como dependencia de desarrollo. Lo buscamos con el asistente y me ofrece lo siguiente:

.

```
Search for a package:
Would you like to define your dev dependencies (require-dev) interactively [yes]? yes
Search for a package: phpunit

Found 15 packages matching phpunit

   [0] phpunit/phpunit
   [1] phpunit/php-timer
   [2] phpunit/php-text-template
   [3] phpunit/php-file-iterator
   [4] phpunit/php-code-coverage
   [5] phpunit/phpunit-mock-objects Abandoned. No replacement was suggested.
   [6] symfony/phpunit-bridge
   [7] jean85/pretty-package-versions
   [8] phpunit/php-invoker
   [9] phpunit/php-token-stream Abandoned. No replacement was suggested.
  [10] johnkary/phpunit-speedtrap
  [11] phpstan/phpstan-phpunit
  [12] brianium/paratest
  [13] yoast/phpunit-polyfills
  [14] spatie/phpunit-snapshot-assertions

Enter package # to add, or the complete package name if it is not listed:
```

El que nos interesa es el primero, que está en el repositorio de phpunit, lo añado. Podemos indicarlo por nombre o por número, en este caso “phpunit/phpunit”, o bien “0”. La configuración es similar a la de los paquetes de proyecto, así que sigo los pasos.

```
Enter package # to add, or the complete package name if it is not listed: 0
Enter the version constraint to require (or leave blank to use the latest version):
Using version ^10.5 for phpunit/phpunit
Search for a package:
```

Una vez tengamos todos los paquetes de desarrollo, hacemos enter para finalizar.

<u>Mapeo de directorios/namespaces en la ruta relativa:</u>

PSR-4 es un estándar de composer para cargar las librerías de forma automática.

La siguiente sección es la carga automática de librerías en nuestra aplicación. Nos pregunta si deseamos añadir algún tipo de namespace o de directorio dentro de **PSR-4** como ruta relativa (por defecto pone **src**) para encontrar dichas librerías.

En versiones iniciales, composer tenía un estándar de autocarga llamado PSR-0. Si nos lo encontramos en algún proyecto, debemos sustituirlo por PSR-4, porque PSR-0 está deprecado y sólo se mantiene por compatibilidad, es probable que por poco tiempo.

Init pone como namespace por defecto el nombre de mi proyecto (empresa\proyecto). Propone el directorio **src/** como ruta relativa a nuestros programas. Como vamos a poner todos los fuentes en src, lo aceptamos con enter.

```
Add PSR-4 autoload mapping? Maps namespace "Marta\Proyecto1" to the entered
relative path. [src/, n to skip]:

{
    "name": "marta/proyecto1",
    "description": "Ejercicio de Composer",
    "type": "project",
    "require": {
        "doctrine/dbal": "^3.7"
    },
    "require-dev": {
        "phpunit/phpunit": "^10.5"
    },
    "autoload": {
        "psr-4": {
            "Marta\\Proyecto1\\": "src/"
        }
    }
}

Do you confirm generation [yes]?
```

Final del asistente: Generación del composer.json

El mapeo era el último paso. Para finalizar nos muestra el fichero composer.json configurado con el esquema definido para composer, y nos pide confirmación de que está bien.
Podemos ver las secciones de librerías (require) y librerías de desarrollo (require-dev)
Si decimos que sí, nos generará el composer.json y la carpeta src vacía.

Instalación de dependencias:

Una vez creados composer.json y src, me pregunta si quiero instalar las dependencias configuradas: dbal y phpunit.

Ya que estamos con el asistente, pues le decimos que sí!!. Esto hará que nos instale todas las dependencias. Se conecta al repositorio, descarga, descomprime e instala en mi proyecto las dependencias indicadas, dentro de la carpeta de proyecto, en un directorio vendor. Al terminar:

```
3 package suggestions were added by new dependencies, use `composer suggest`
 to see details.
Generating autoload files
26 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
No security vulnerability advisories found.
PSR-4 autoloading configured. Use "namespace Marta\Proyecto1;" in src/
Include the Composer autoloader with: require 'vendor/autoload.php';
```

Dentro de vendor podemos ver los paquetes instalados:

```
alumno@alumno-VirtualBox:~/proyectos/vendor$ ls -l
total 44
-rw-rw-r--  1 alumno alumno  771 dic 13 12:38 autoload.php
drwxrwxr-x  2 alumno alumno 4096 dic 13 12:38 bin
drwxrwxr-x  2 alumno alumno 4096 dic 13 12:38 composer
drwxrwxr-x  6 alumno alumno 4096 dic 13 12:38 doctrine
drwxrwxr-x  3 alumno alumno 4096 dic 13 12:38 myclabs
drwxrwxr-x  3 alumno alumno 4096 dic 13 12:38 nikic
drwxrwxr-x  4 alumno alumno 4096 dic 13 12:38 phar-io
drwxrwxr-x  8 alumno alumno 4096 dic 13 12:38 phpunit
drwxrwxr-x  4 alumno alumno 4096 dic 13 12:38 psr
drwxrwxr-x 17 alumno alumno 4096 dic 13 12:38 sebastian
drwxrwxr-x  3 alumno alumno 4096 dic 13 12:38 theseer
alumno@alumno-VirtualBox:~/proyectos/vendor$
```

Así, una vez finalizado init, la estructura creada (en el directorio donde ejecuté el comando (composer init) es la siguiente:.

```
alumno@alumno-VirtualBox:~/proyectos$ ls -l
total 88
-rw-rw-r--  1 alumno alumno   315 dic 13 12:38 composer.json
-rw-rw-r--  1 alumno alumno 75289 dic 13 12:38 composer.lock
drwxrwxr-x  2 alumno alumno  4096 dic 13 12:38 src
drwxrwxr-x 12 alumno alumno  4096 dic 13 12:38 vendor
alumno@alumno-VirtualBox:~/proyectos$
```

1. Los ficheros composer.json y composer.lock con la configuración del proyecto
2. La carpeta src, inicialmente vacía, donde pondré mis fuentes
3. La carpeta vendor, que contiene:
    a. Los ficheros autoload.php (carga de librerías)
    b. Varias carpetas con las librerías iniciales del proyecto

COMANDO CREATE PROJECT

Este comando es otra forma, alternativa al INIT, para crear un proyecto en Composer
Lo que hace es crear un proyecto **a partir de un paquete existente**, algo así como un “git clone svn” (clonación de repositorios de subversion).

Imaginemos que queremos crear un proyecto PHP con el framework Symfony. En la página de symfony nos indica cómo hacerlo con composer:

https://symfony.com/doc/current/setup.html

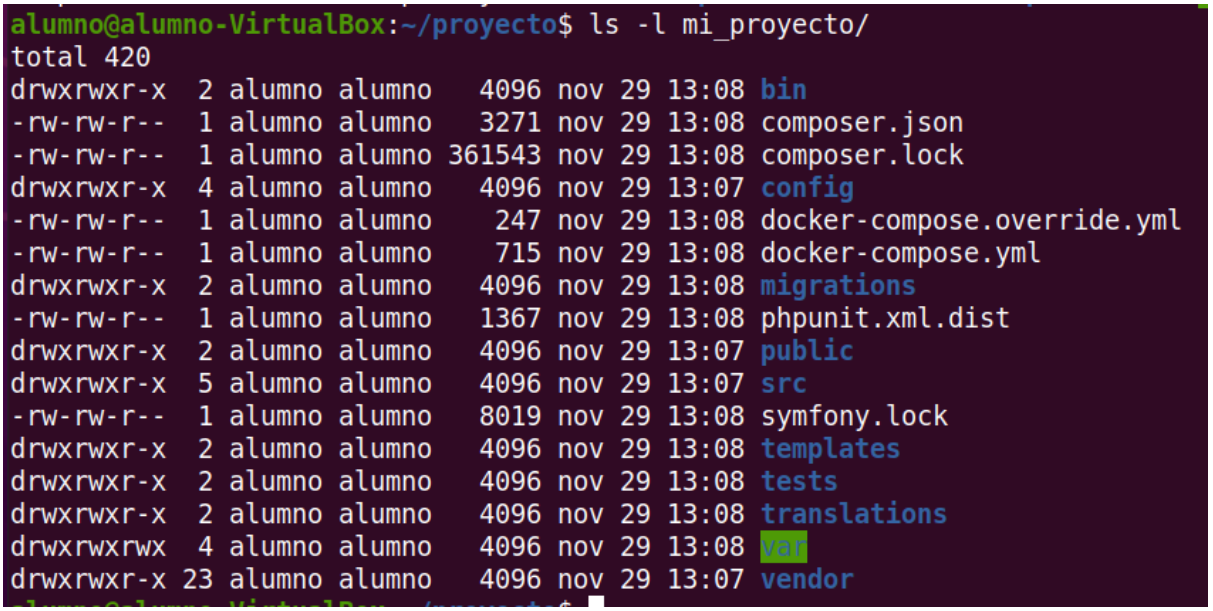
- Para un proyecto web:

$ composer create-project symfony/website-skeleton nombre_proyecto

- Para un microservicio, API o aplicación de consola:
  $ composer create-project symfony/skeleton nombre_proyecto

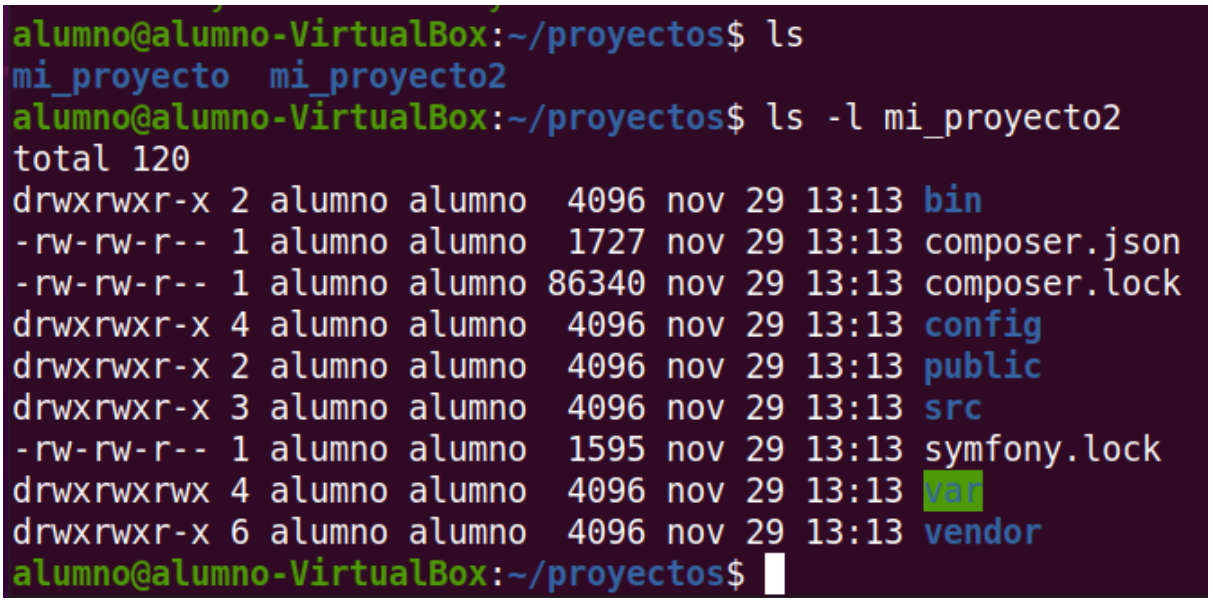
Si ejecuto la primera opción, tras un buen rato de instalación, crea la carpeta del proyecto e instala dentro toda la estructura necesaria para trabajar con symfony en un proyecto web.

$ composer create-project symfony/website-skeleton mi_proyecto



Si ejecuto el segundo, crea la estructura para un proyecto tipo API, o microservicio

$ composer create-project symfony/skeleton mi_proyecto2



Vamos a trabajar en mi_proyecto2. Veremos que ha generado automáticamente el composer.json con todas las dependencias de symfony.
Podríamos crear un repositorio de código con git, a partir de este proyecto.

Si visualizamos el composer.json de symfony, tiene licencia propietaria, es estable, e indica que prefiere extensiones estables (no betas).
No hay ninguna dependencia de desarrollo. Vamos a ver las dependencias instaladas


```
"type": "project",
"license": "proprietary",
"minimum-stability": "stable",
"prefer-stable": true,
"require": {
    "php": ">=8.1",
    "ext-ctype": "*",
    "ext-iconv": "*",
    "doctrine/doctrine-bundle": "^2.11",
    "doctrine/doctrine-migrations-bundle": "^3.3",
    "doctrine/orm": "^2.17",
    "phpdocumentor/reflection-docblock": "^5.3",
    "phpstan/phpdoc-parser": "^1.24",
    "symfony/asset": "6.1.*",
    "symfony/asset-mapper": "6.1.*",
    "symfony/console": "6.1.*",
    "symfony/doctrine-messenger": "6.1.*",
    "symfony/dotenv": "6.1.*",
    "symfony/expression-language": "6.1.*",
    "symfony/flex": "^2",
    "symfony/form": "6.1.*",
    "symfony/framework-bundle": "6.1.*",
    "symfony/http-client": "6.1.*",
    "symfony/intl": "6.1.*",
    "symfony/mailer": "6.1.*",
    "symfony/mime": "6.1.*",
    "symfony/monolog-bundle": "^3.0",
    "symfony/notifier": "6.1.*",
    "symfony/process": "6.1.*",
    "symfony/property-access": "6.1.*",
    "symfony/property-info": "6.1.*",
    "symfony/runtime": "6.1.*",
    "symfony/security-bundle": "6.1.*",
    "symfony/serializer": "6.1.*",
    "symfony/stimulus-bundle": "^2.13",
    "symfony/string": "6.1.*",
    "symfony/translation": "6.1.*",
    "symfony/twig-bundle": "6.1.*",
    "symfony/ux-turbo": "^2.13",
    "symfony/validator": "6.1.*",
    "symfony/web-link": "6.1.*",
    "symfony/yaml": "6.1.*",
    "twig/extra-bundle": "^2.12|^3.0",
    "twig/twig": "^2.12|^3.0"
},
```

Dentro del mundo de Composer, hay algunas dependencias virtuales: Indican qué versión de php son la mínima para poder ejecutar este proyecto. >= indica un mínimo de 8.1 de PHP. Son restricciones. Con * indicamos que necesitamos esa librería para correr el proyecto,indicando que cualquier versión.

```
"config": {
    "allow-plugins": {
        "composer/package-versions-deprecated": true,
        "symfony/flex": true,
        "symfony/runtime": true
    },
    "optimize-autoloader": true,
    "preferred-install": {
        "*": "dist"
    },
    "sort-packages": true
},
```

Indica que se optimice la autocarga. La instalación, preferentemente, será desde dist (descarga en formato zip, luego descomprime e instala). Hay otras.

```
"autoload": {
    "psr-4": {
        "App\\": "src/"
    }
},
"autoload-dev": {
    "psr-4": {
        "App\\Tests\\": "tests/"
    }
},
```

El autoload en desarrollo es distinto que en producción

COMANDOS INSTALL Y UPDATE

Vamos a ver cómo instalar paquetes sin el init.

El archivo **composer.lock**, es un json con información extendida, referencias de las dependencias. La versión de dependencias de un proyecto queda bloqueada. Este archivo tiene que estar, junto a composer.json, en GIT.

El composer.lock almacena la versión exacta de lo que instalamos. Si escribimos composer install, instala lo que encuentra en el composer.lock, es decir, la versión exacta de las dependencias tal y como están definidas en el fichero.

Si lo que queremos es actualizar las versiones de mis dependencias a otras posteriores, utilizaré el comando composer update, que no sólo instalará las nuevas versiones, sino que también actualizará el composer.lock.

Así:

- **Install:** Instala todo lo que hay definido en composer.lock y no esté ya instalado. Versión exacta.
- **Update:** Instala lo que no esté instalado de composer lock. Si hay nuevas versiones, descarga e instala las nuevas, sustituyendo a las antiguas (instaladas o no). Finalmente actualiza composer.lock para indicar cuáles son las versiones actuales de nuestro proyecto. Por eso es tan importante que composer.lock y composer.json estén en GIT, para mantener sus versiones.

Por ello, sólo utilizaré update en los casos estrictamente necesarios, consensuados con el equipo de trabajo, que deberá actualizar también sus dependencias.

NOTA IMPORTANTE: Después de instalar nuevas librerías, es necesario hacer un update, para actualizar todas las dependencias.

Paso 1: **Borramos** la carpeta vendor, con el objetivo de hacer la instalación de paquetes por comando en vez de con el asistente del init.

Paso 2: Ejecución de **composer install**: Como habíamos borrado vendor, composer vuelve a crearlo y a instalar la versión exacta de las dependencias que había en **composer.lock**.

```
alumno@alumno-VirtualBox:~$ composer install
Installing dependencies from lock file (including require-dev)
Verifying lock file contents can be installed on current platform.
Package operations: 34 installs, 0 updates, 0 removals
  - Installing psr/log (3.0.0): Extracting archive
  - Installing psr/cache (3.0.0): Extracting archive
  - Installing doctrine/event-manager (2.0.0): Extracting archive
  - Installing doctrine/deprecations (v1.0.0): Extracting archive
  - Installing doctrine/cache (2.2.0): Extracting archive
  - Installing doctrine/dbal (3.5.1): Extracting archive
  - Installing sebastian/version (3.0.2): Extracting archive
  - Installing sebastian/type (3.2.0): Extracting archive
  - Installing sebastian/resource-operations (3.0.3): Extracting archive
  - Installing sebastian/recursion-context (4.0.4): Extracting archive
  - Installing sebastian/object-reflector (2.0.4): Extracting archive
  - Installing sebastian/object-enumerator (4.0.4): Extracting archive
  - Installing sebastian/global-state (5.0.5): Extracting archive
  - Installing sebastian/exporter (4.0.5): Extracting archive
  - Installing sebastian/environment (5.1.4): Extracting archive
  - Installing sebastian/diff (4.0.4): Extracting archive
  - Installing sebastian/comparator (4.0.8): Extracting archive
  - Installing sebastian/code-unit (1.0.8): Extracting archive
```

Paso 3: Ejecución de **composer update**: Compara el composer.lock con las versiones actuales, y actualiza a la última versión, sustituyendo si existía e instalando si no existía. Actualizará el **composer.lock** para que figuren las nuevas versiones.

```
alumno@alumno-VirtualBox:~$ composer update
Loading composer repositories with package information
Info from https://repo.packagist.org: #StandWithUkraine
Updating dependencies
Nothing to modify in lock file
Installing dependencies from lock file (including require-dev)
Nothing to install, update or remove
Generating autoload files
28 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
No security vulnerability advisories found
alumno@alumno-VirtualBox:~$
```

COMANDOS REQUIRE y REMOVE

Permiten añadir-quitar dependencias de composer.

Veamos nuestro composer.json:

```
{
    "name": "marta/proyecto1",
    "description": "Ejercicio de Composer",
    "type": "project",
    "require": {
        "doctrine/dbal": "^3.7"
    },
    "require-dev": {
        "phpunit/phpunit": "^10.5"
    },
    "autoload": {
        "psr-4": {
            "Marta\\Proyecto1\\": "src/"
        }
    }
}
```

Para quitar la dependencia doctrine/dbal, escribimos:

$ composer remove doctrine/dbal

```
alumno@alumno-VirtualBox:~/proyectos$ composer remove doctrine/dbal
./composer.json has been updated
Running composer update doctrine/dbal
Loading composer repositories with package information
Updating dependencies
Lock file operations: 0 installs, 0 updates, 6 removals
  - Removing doctrine/cache (2.2.0)
  - Removing doctrine/dbal (3.7.2)
  - Removing doctrine/deprecations (1.1.2)
  - Removing doctrine/event-manager (2.0.0)
  - Removing psr/cache (3.0.0)
  - Removing psr/log (3.0.0)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 0 installs, 0 updates, 6 removals
  - Removing psr/log (3.0.0)
  - Removing psr/cache (3.0.0)
  - Removing doctrine/event-manager (2.0.0)
  - Removing doctrine/deprecations (1.1.2)
  - Removing doctrine/dbal (3.7.2)
  - Removing doctrine/cache (2.2.0)
Generating autoload files
23 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
No security vulnerability advisories found.
alumno@alumno-VirtualBox:~/proyectos$
```

Elimina dbal. Como vemos, también actualiza el **composer.json** y el **composer.lock** (lo primero que nos dice!!).
También elimina el directorio de la carpeta vendor.
Composer.json quedará:

```
{
    "name": "marta/proyecto1",
    "description": "Ejercicio de Composer",
    "type": "project",
    "require-dev": {
        "phpunit/phpunit": "^10.5"
    },
    "autoload": {
        "psr-4": {
            "Marta\\Proyecto1\\": "src/"
        }
    }
}
```

Ya no tiene require, pues no hay ninguna dependencia.

Para instalar una nueva dependencia, utilizo el comando require:

```
$ composer require doctrine/dbal
```

```
alumno@alumno-VirtualBox:~/proyectos$ composer require doctrine/dbal
./composer.json has been updated
Running composer update doctrine/dbal
Loading composer repositories with package information
Updating dependencies
Lock file operations: 6 installs, 0 updates, 0 removals
  - Locking doctrine/cache (2.2.0)
  - Locking doctrine/dbal (3.7.2)
  - Locking doctrine/deprecations (1.1.2)
  - Locking doctrine/event-manager (2.0.0)
  - Locking psr/cache (3.0.0)
  - Locking psr/log (3.0.0)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 6 installs, 0 updates, 0 removals
  - Installing psr/log (3.0.0): Extracting archive
  - Installing psr/cache (3.0.0): Extracting archive
  - Installing doctrine/event-manager (2.0.0): Extracting archive
  - Installing doctrine/deprecations (1.1.2): Extracting archive
  - Installing doctrine/cache (2.2.0): Extracting archive
  - Installing doctrine/dbal (3.7.2): Extracting archive
1 package suggestions were added by new dependencies, use `composer suggest` to
see details.
Generating autoload files
26 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
No security vulnerability advisories found.
Using version ^3.7 for doctrine/dbal
```

Y vuelve a cambiar **composer.json** para que figure la nueva dependencia.

```
{
    "name": "marta/proyecto1",
    "description": "Ejercicio de Composer",
    "type": "project",
    "require-dev": {
        "phpunit/phpunit": "^10.5"
    },
    "autoload": {
        "psr-4": {
            "Marta\\Proyecto1\\": "src/"
        }
    },
    "require": {
        "doctrine/dbal": "^3.7"
    }
}
```

Para eliminar o instalar una **dependencia de desarrollo**, utilizaremos el flag --dev:

$ composer require --dev phpunit/phpunit

Y luego actualizará composer.json

COMANDO SEARCH

Busca paquetes, pero hay que decirle algo, al menos un nombre o un prefijo.

Por ejemplo, si busco los paquetes de symfony:

$ composer search symfony

Me retornará el resultado de la búsqueda, para que yo elija el paquete:

```
alumno@alumno-VirtualBox:~$ composer search symfony
symfony/var-dumper                  Provides mechanisms for walking through any arbitrary PHP variable
symfony/translation                 Provides tools to internationalize your application
symfony/routing                     Maps an HTTP request to a set of configuration variables
symfony/process                     Executes commands in sub-processes
symfony/polyfill-php72              Symfony polyfill backporting some PHP 7.2+ features to lower PHP ve...
symfony/polyfill-mbstring           Symfony polyfill for the Mbstring extension
symfony/polyfill-intl-idn           Symfony polyfill for intl's idn_to_ascii and idn_to_utf8 functions
symfony/polyfill-ctype              Symfony polyfill for ctype functions
symfony/http-kernel                 Provides a structured process for converting a Request into a Response
symfony/http-foundation             Defines an object-oriented layer for the HTTP specification
symfony/finder                      Finds files and directories via an intuitive fluent interface
symfony/filesystem                  Provides basic utilities for the filesystem
symfony/event-dispatcher-contracts  Generic abstractions related to dispatching event
symfony/event-dispatcher            Provides tools that allow your application components to communicat...
symfony/css-selector                Converts CSS selectors to XPath expressions
alumno@alumno-VirtualBox:~$
```

Los paquetes para composer pueden buscarse con este comando, o bien directamente en un repositorio en la web, por ejemplo el repositorio La Liga de los paquetes extraordinarios:

https://thephpleague.com/

Este repositorio es popular. Puedo conectarme a su web y hacer la búsqueda por ahí, o puedo escribirlo como parámetro del comando search:

```
alumno@alumno-VirtualBox:~$ composer search phpleague
league/oauth2-google                        Google OAuth 2.0 Client Provider for The PHP League OAuth2-C...
league/oauth2-facebook                      Facebook OAuth 2.0 Client Provider for The PHP League OAuth2...
thenetworg/oauth2-azure                     Azure Active Directory OAuth 2.0 Client Provider for The PHP...
steverhoades/oauth2-openid-connect-server   An OpenID Connect Server that sites on The PHP League's OAut...
stevenmaguire/oauth2-keycloak               Keycloak OAuth 2.0 Client Provider for The PHP League OAuth2...
league/oauth2-linkedin                      LinkedIn OAuth 2.0 Client Provider for The PHP League OAuth2...
wohali/oauth2-discord-new                   Discord OAuth 2.0 Client Provider for The PHP League OAuth2-...
stevenmaguire/oauth2-salesforce             Salesforce OAuth 2.0 Client Provider for The PHP League OAut...
stevenmaguire/oauth2-microsoft              Microsoft OAuth 2.0 Client Provider for The PHP League OAuth...
riskio/oauth2-auth0                         Auth0 OAuth 2.0 Client Provider for The PHP League OAuth2-Cl...
patrickbussmann/oauth2-apple                Sign in with Apple OAuth 2.0 Client Provider for The PHP Lea...
league/oauth2-instagram                     Instagram OAuth 2.0 Client Provider for The PHP League OAuth...
league/oauth2-github                        Github OAuth 2.0 Client Provider for The PHP League OAuth2-C...
craftcms/oauth2-craftid                     Craft OAuth 2.0 Client Provider for The PHP League OAuth2-Cl...
calcinai/oauth2-xero                        Xero OAuth 2.0 Client Provider for The PHP League OAuth2-Client
alumno@alumno-VirtualBox:~$
```

Como veremos, hay paquetes de oauth (open authentication) con google, con facebook, con azure, linkedin, discord, instagram, etc.

El repositorio de paquetes por defecto de composer es Packagist. Es de ahí de donde composer obtiene la información para gestionar las dependencias. Su web es :

https://packagist.org/

Esta web permite buscar paquetes, de forma más gráfica y completa que el search modo comando de composer.

Si entramos en un paquete, nos dice quién mantiene el paquete, las dependencias, los requisitos de versiones, con qué paquetes entra en conflicto, etc.

COMANDO VALIDATE

Nos permite validar el manifiesto composer.json con su esquema. Si lo modificamos a mano, comprueba la sintaxis contra el esquema definido para composer.json, y nos indica el fallo de sintaxis concreto: Una llave sin cerrar, entradas no definidas, etc.

COMANDO SELF-UPDATE

Comprueba la versión de composer, y permite actualizarla.

COMANDO DUMPAUTOLOAD

```
$ composer dumpautoload
```

Actualiza una lista, la que contiene todos los programas/librerías/clases que deben ser cargados con un include en mi proyecto.
Si nuestro proyecto tiene muchos ficheros, y es lo habitual, hacer include de todos ellos es una labor tediosa. Con composer no necesito hacer include o require de los programas o librerías que necesitan. Con composer se cargan (incluyen) de forma automática con un "autoloader", configurado en el composer.json, donde se pone la carpeta donde encontrar las librerías, y el "cargador", en nuestro ejemplo el PSR4, aunque hay otros.
Un artículo interesante acerca de diversos autoloaders es el siguiente:
https://jinoantony.com/blog/how-composer-autoloads-php-files

Así, al ejecutar composer dumpautoload, se optimiza la gestión de la autocarga (includes y requires automáticos).

```
alumno@alumno-VirtualBox:~/proyectos/m$ composer dumpautoload
Generating autoload files
Generated autoload files
alumno@alumno-VirtualBox:~/proyectos/m$
```

Si no existe, se genera el fichero vendor/autoload.php, con el código a ejecutar para la correcta inclusión de las librerías en mi proyecto.

Este comando no reinstala ni actualiza nada, genera el **autoload.php**, y comprueba de nuevo todo lo que hay para cargar las clases que puedan no estar incluidas en el proyecto.

CREANDO NUESTROS PROPIOS COMANDOS:

Podemos crear nuestros propios comandos para ejecutar con Composer. Se configuran en el fichero composer.json.

En composer.json existe la posibilidad de añadir una clave “scripts”. Puedo modificarlo y añadir mis propios comandos personalizados, dentro de esa clave “scripts”.

Paso 1: Añado un comando “lista”, para que cuando lo escriban haga “ls -la”:

```
“scripts”: {
      “lista” : “ls -la”
}
```

```
{
    "name": "marta/proyecto1",
    "description": "Ejercicio de Composer",
    "type": "project",
    "autoload": {
        "psr-4": {
            "Marta\\Proyecto1\\": "src/"
        }
    },
    "require": {
        "doctrine/dbal": "^3.7"
    },
    "require-dev": {
        "phpunit/phpunit": "^10.5"
    },
    "scripts": {
        "lista": "ls -la"
    }
}
```

Paso 2: Compruebo que lo he escrito todo bien, con validate

```
    "scripts": {
        "lista": "ls -la"
    }
}
alumno@alumno-VirtualBox:~$ composer validate composer.json
composer.json is valid, but with a few warnings
See https://getcomposer.org/doc/04-schema.md for details on the schema
# General warnings
- No license specified, it is recommended to do so. For closed-source software you may use "proprietary" as
 license.
```

Paso 3: Una vez comprobado que he modificado bien el composer.json, ejecuto el nuevo comando:

$ composer lista

A partir de este momento, podré ejecutar el script desde el directorio raíz del proyecto, que es donde está composer.json.

```
alumno@alumno-VirtualBox:~/proyectos$ composer lista
> ls -la
total 96
drwxrwxr-x  4 alumno alumno  4096 dic 14 16:15 .
drwxr-x--- 24 alumno alumno  4096 dic 14 15:37 ..
-rw-rw-r--  1 alumno alumno   365 dic 14 16:15 composer.json
-rw-rw-r--  1 alumno alumno 75289 dic 14 16:05 composer.lock
drwxrwxr-x  2 alumno alumno  4096 dic 13 12:38 src
drwxrwxr-x 12 alumno alumno  4096 dic 14 16:05 vendor
alumno@alumno-VirtualBox:~/proyectos$
```

Si quiero ejecutarlo desde otro directorio, composer lanza un warning pidiendo confirmación.

```
alumno@alumno-VirtualBox:~/proyectos/vendor$ composer lista
No composer.json in current directory, do you want to use the one at /home/alumn
o/proyectos? [Y,n]? y
Always want to use the parent dir? Use "composer config --global use-parent-dir
true" to change the default.
> ls -la
total 96
drwxrwxr-x  4 alumno alumno  4096 dic 14 16:15 .
drwxr-x--- 24 alumno alumno  4096 dic 14 15:37 ..
-rw-rw-r--  1 alumno alumno   365 dic 14 16:15 composer.json
-rw-rw-r--  1 alumno alumno 75289 dic 14 16:05 composer.lock
drwxrwxr-x  2 alumno alumno  4096 dic 13 12:38 src
drwxrwxr-x 12 alumno alumno  4096 dic 14 16:05 vendor
alumno@alumno-VirtualBox:~/proyectos/vendor$
```

Si observamos los mensajes, me permite ejecutarlo si confirmo, pero lo ejecuta… *en el directorio raíz del proyecto!!*

También me indica que puedo modificar la configuración de composer para que no sea necesaria la confirmación.

Ejemplo con 2 scripts:

Voy a añadir un segundo script a mi composer.json. Este script ejecutará phpunit (paquete para pruebas unitarias), que está en vendor/bin

Si ahora ejecuto

$composer pruebas

ejecutará el comando phpunit. El script se lanza desde el raiz de mi proyecto.

Esto es muy útil, pues podré diseñar comandos de composer para ejecutar mis propios scripts cómodos, como por ejemplo tirar test unitarios concretos, etc.

Puedo añadir tantos comandos como quiera dentro de esa sección script del composer.json