PHP 10권 – 패키지 설치

**06. 패키지 설치**

====================

컴포저의 패키지는 JSON형태의 데이터를 가지는 composer.json 파일에서 통합 관리됩니다. 패키지 들은 JSON형식에 따라 “패키지이름”:”버전” 형태로 저장이 됩니다.

컴포저를 사용하기 위해서는 composer.json 파일을 프로젝트 루투에 작성을 합니다. 이 파일은 외부 라이브러리의 의존성에 대해서 적혀 있습니다.

**06.1 컴포저 초기화**

처음 컴포저를 시작하는 경우에는 컴포저를 초기화 해주어야 합니다.

**06.1.1 명령어 방식**

컴포저 초기화는 터미널 창에서 다음과 같이 명령을 입력합니다.

|  |
| --- |
| composer **init** |

초기화 명령을 입력하면 몇 단계입력을 요구합니다. 입력된 내용을 기반으로 컴포저 관리파일인 composer.json 파일을 자동으로 생성해 줍니다.

|  |
| --- |
| d:\htdocs\composer>**composer init**  Welcome to the Composer config generator  This command will guide you through creating your composer.json config.  Package name (<vendor>/<name>) [infoh/composer]:  Description []:  Author [, n to skip]: HojinLee <infohojin@naver.com>  Minimum Stability []:  Package Type (e.g. library, project, metapackage, composer-plugin) []:  License []:  Define your dependencies.  Would you like to define your dependencies (require) interactively [yes]? no  Would you like to define your dev dependencies (require-dev) interactively [yes]? no  {  "name": "infoh/composer",  "authors": [  {  "name": "HojinLee",  "email": "infohojin@naver.com"  }  ],  "require": {}  }  Do you confirm generation [yes]? yes |

입력이 모두 완료가 되면 composer.json 파일이 하나 자동으로 생성되게 됩니다. 이 파일은 JSON 형태로 작성이 되어 있습니다.

Composer.json 파일에 필요한 패키지와 정보를 작성하면 자동으로 관련 페키지를 설치하고, lock 파일을 생성을 합니다.

**06.1.2 직접작성**

만일 텍스트 에디터로 설정파일을 작성하는데 능숙하다면 초기화 명령을 사용하지않고 하셔도 됩니다.

**06.1.3 composer.json**

초기화 작업으로 처음으로 생성된 composer.json 파일을 한번 살펴 보도록 합니다.

|  |
| --- |
| 파일 | **composer.json** |
| {  "name": "infoh/composer",  "authors": [  {  "name": "HojinLee",  "email": "infohojin@naver.com"  }  ],  "require": {}  } |

**06.2 패키지 디렉토리**

컴포저는 설치 관리되는 패키지를 별도의 특별한 디렉토리에서 관리를 합니다. 컴포저를 이용하여 패키지를 설치하게 되면, 자동적으로 /vendor 디렉토리를 만들게 됩니다.

**06.2.1 Vendor 디렉토리**

페키지들의 충돌을 방지하지 위하여 벤더명을 같이 지정을 하게 됩니다. 벤더명은 회사의 이름 또는 자신의 아이디가 될 수도 있습니다. 벤더명은 중복될 수 없으며 다은 사람이 이미 사용을 하고 있다면, 충돌 되지 않은 이름으로 사용을 해야 합니다.

**06.2.2 네임스페이스**

벤더명은 PHP의 네임스페이스의 첫번째 인자로 사용을 합니다. 네임스페이스가 더 큰 개념으로 클래스의 충돌을 방지하여 위하여 사용을 하는 목적과 부합하게 이름을 벤더명으로 구분을 하는 것입니다.

Vendor 디렉토리는 벤더명으로 서브 폴더를 만들어 각각의 페키지 라이브러리를 관리하게 됩니다.

**06.2.3 번더명 예제**

벤더명과 프로젝트 명을 같이 사용을 할 수도 있습니다.

예제

{

“require”: {

“jiny/jiny”: “1.2.\*”

}

}

**06.3 페키지 설치**

컴포저를 이용하여 페키지를 설치하는 방법에는 2가지가 있습니다. Require 명령을 이용하는 방법과 install 명령을 이용하는 방법입니다.

**06.3.1 Require**

Require 명령은 새로운 패키지를 검색하여 설치하고, composer.json 파일을 갱신하게 됩니다. 즉, 프로젝트를 진행하게 되면서 처음으로 페키지를 설치해야 하는 경우 매우 유용합니다.

**06.3.2 Install**

Install 명령은 composer.json 파일에 나열되어 있는 패키지 목록을 찾아서 새로운 프로젝트에 기본 페키지들을 설치할 때 유용합니다. 프로젝트를 진행하고 있는 도중에 새로운 다른 프로젝트를 시작할려고 할 때, 기존 페키지의 목록과 동일하게 시작하고 싶다고 한다면 install 방식은 매우 유용할 것입니다.

또한, 응용프로그램을 배포할 때 composer.json 파일만 제고하고, 관련 페키지들은 각자 설치하여 사용하게 할 수도 있습니다.

설치할 패지키 목록이 들어 있는 composer.json 파일을 가지고 있다면, 쉽게 관련된 페키지를 설치할 수 있습니다.

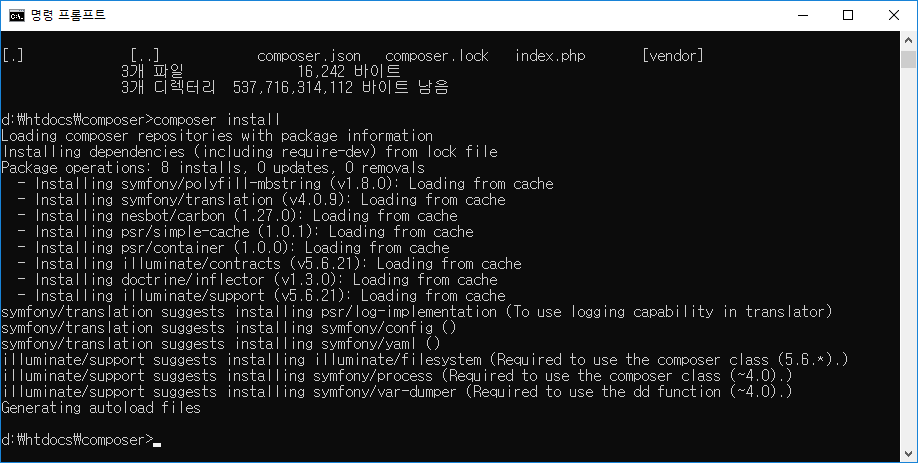
composer install

위의 방식은 이전의 require를 이용하여 페지키를 설치하는 것과 유사합니다. Require는 페키지를 찾아 설치하고 json 파일을 갱신하게 됩니다. 하지만 Install은 json 파일에 있는 목록을 찾아 설치해 주는 것입니다.

실습을 해보기 위해서 먼저 vendor 폴더의 내용을 삭제해 봅니다.

그리고 composer install

을 하게 되면 패키지들이 다시 설치된 모습을 확인해 보실 수 있습니다.



**06.3.3 Require-dev**

Require-dev를 설치할 수 있습니다.

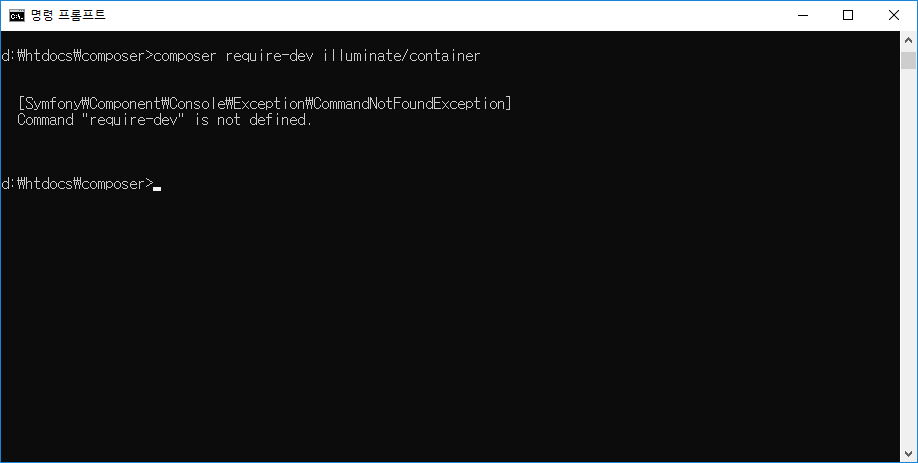
composer require --dev 페키지명

아래와 같이 페키지를 require-dev 쪽으로 설치할 수 있습니다.

d:\htdocs\composer>composer require --dev illuminate/container

[Symfony\Component\Console\Exception\CommandNotFoundException]

Command "require-dev" is not defined.



설치후에 다시 composer.json 파일을 확인해 보도록 합니다.

{

"name": "infoh/composer",

"authors": [

{

"name": "HojinLee",

"email": "infohojin@naver.com"

}

],

"require": {

"illuminate/support": "^5.6"

}**,**

**"require-dev": {**

**"illuminate/container": "^5.6"**

**}**

}

굵은 표시로 된 영역이 자동적으로 추가된 것을 확인을 할 수 있습니다. 이렇게 설치된 패키지는 컴포저 오토로드가 아직 알지 못합니다. 내부적인 vendor/composer 파일들을 재작성 해주어야 합니다. 재작성하는 방법으로는 **dumpautoload 명령**을 수행하는 것입니다.

**06.4 Dump Autoload**

**Dumpautoload**

패키지를 추가하게 되면 갱신이 필요하게 됩니다. 터미널 창에서 다음과 같이 명령을 입력합니다.

composer dumpautoload

매번 패키지를 재설치하는 경우 내부적으로 컴포저가 인식할 수 있도록 dumpautoload를 실행해 주는 것이 좋습니다.

**06.5 Create Project**

컴포저의 create-project 명령은 기존 install이나 require 명령과는 다소 차이점이 있습니다.

Install/require는 페키지를 /vendor 안에 설치하여 관리를 하지만, create-project 명령은 페키지를 현재의 프로젝트에 폴더구조를 만들어 설치하게 됩니다.

이는 페키지를 다운로드하는 것이 아닌 새로운 내용을 담을 페키지를 생성을 하는 명령어 입니다.

**$ composer create-project 패키지명 폴더명 버전**

예를 들어 라라벨과 같은 프레임워크는 설치파일이 컴포저를 통하여 배포됩니다.

$ composer create-project laravel/laravel

과 같이 입력하게 되면 컴포저를 통하여 설치파일과 디렉토리가 생성됩니다.

이는 약간 git의 clone과도 유사하게 생각할 수 있습니다. Git의 clone으로 설치하게 되면 소스 형상관리 형태로 복제되는 것과 다릅니다.

실제적인 파일만 저장하게 됩니다.

**06.5.2 지니 프레임웍크 설치**

예를들어 필자가 만들고 있는 지니 프레임웍크를 설치해 보도록 합니다.

**06.6 패키지 버전**

라이브러리의 패키지 버전을 지정하고 관리하는 것은 의존성을 유지하는데 매우 유용한 방법입니다.

컴포저에서 버전을 지정하는 몇 개의 연산자를 가지고 있습니다.

**06.6.1 ~ 연산자**

**06.6.2 ^ 연산자**

**06.6.3 \* 연산자**

**06.7 패키지 갱신**

컴포저와 페키지를 설치하여 사용하고 있는 중간에, 일부 패키지가 버전업이 되었다고 생각해 봅시다.

그럼 일부 갱신된 패키지들의 버전을 일일이 확인해서 재설치 해주는 것은 매우 힘든 일 일것입니다.

컴포저를 이용하면 쉽게 사용중인 페키지의 버전을 확인하여 업데이트를 할 수 있습니다.

|  |
| --- |
| composer update |

Update는 Require를 통하여 설치하는 것 대신에, 직접 composer.json 파일을 수정후에 update명령어로 설치를 할 수도 있습니다.

**06.8 패키지 삭제**

설치한 페키지를 삭제할 수 있습니다. 터미널창에서 간단하게 remove 명령만 입력하게 되면 설치한 페키지를 쉽게 제거할 수 있습니다.

composer remove 패키지이름

페키지를 제거하게 되면 관련 파일들을 자동으로 삭제함과 동시에 composer.json 파일도 갱신해 주기 때문에 매우 편리합니다.

**06.9 의존성 유지**

프로젝트를 여러 개발자들과 협업을 하여 진행할 경우, 각각의 환경설정을 동일 하게 유지하는 것은 매우 중요합니다.

그래서 컴포저는 의존성의 정보를 기록한 composer.lock 파일을 자동으로 생성해 주게 됩니다. composer.lock 파일을 개발자들과 공유하여 팀원들 간의 의존성을 일치할 수 있습니다.

**06.9.1 install**

컴포저의 install 명령은 composer.json 파일의 내용을 기준으로 새롭게 패키지들을 다운로드 받아 설치를 하게 됩니다.

하지만 composer.lock 파일 있을 경우 composet.json 파일대신에 lock 파일을 읽어서 페키지를 설치하게 됩니다.

**06.9.2 update**

업데이트는 composer.json 파일을 적요하여 최신 패키지의 버전을 설치합니다. 설치후에 새로운 composer.lock 파일을 생성을 하게 됩니다.

**06.9.3 lock 파일의 배포**

이처럼 composer.lock 파일은 개발 팀원간에 패키지의 의존성을 일치하는데 매우 유용한 방법입니다.

깃과 같은 형상관리 소프트웨어를 사용하고 있다면은, composer.loc 파일을 같이 버전관리하여 배포하는 것도 좋은 방법입니다.