생활코딩 PHP 사용자 12월 모임

PUG!!

modern
php user group

COMPOSER

Why & What

쓰면 뭐가 좋나(with 시연)

궁금한 거

그림 출처 : 하상욱 단편시집 '서울시'

Why & What

쓰면 뭐가 좋나(with 시연)

궁금한 거

There is a huge trend of
reinventing the wheel over and over
in the PHP world.

출처 : Composer: Part 1 – What & Why

The lack of a package manager means that every library author has an incentive **not** to use any other library, otherwise users end up in dependency hell when they want to install it.

출처 : Composer: Part 1 – What & Why

Composer is
the command-line utility with which you install packages.

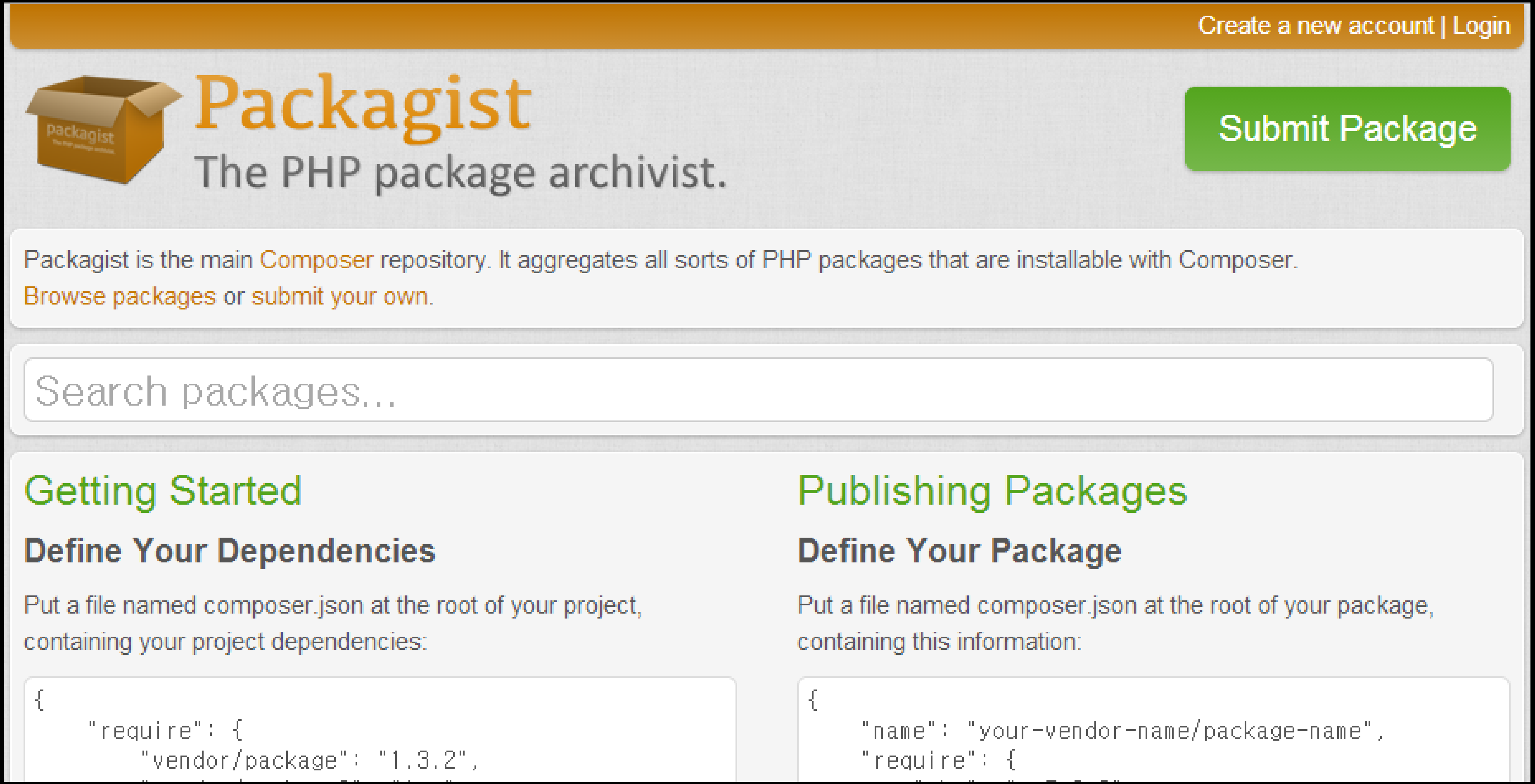
Composer는 패키지들을 인스톨하는 커맨드라인 유틸리티이다.

Composer가 해결한 문제

- 프로젝트가 여러 개의 라이브러리에 의존적이다
- 몇몇 라이브러리가 다른 라이브러리에 의존성이 있다
- 무엇에 의존성이 있는지 선언할 수 있다
- Composer는 설치할 필요가 있는 패키지 버전을 찾아 설치한다. (프로젝트 안으로 설치한다는 뜻이다)

번역 출처 : PHP 의존성 관리도구 – Composer 시작하기 http://haruair.com/blog/1860

Packagist is the default package repository.



Packagist is the main Composer repository. It aggregates all sorts of PHP packages that are installable with Composer.
Browse packages or submit your own.

Search packages...

Getting Started

Define Your Dependencies

Put a file named composer.json at the root of your project, containing your project dependencies:

```
{
    "require": {
        "vendor/package": "1.3.2",
```

Publishing Packages

Define Your Package

Put a file named composer.json at the root of your package, containing this information:

```
{
    "name": "your-vendor-name/package-name",
    "require": {
```

Why & What

쓰면 뭐가 좋나(with 시연)

궁금한 거

자동 설치

의존 관계에 있는 패키지들을 알아서 다운로드 해준다.

Composer install
Composer update

Autoloading

일일이 require, include 하지 않고
composer가 생성해준 autoload.php 하나만 로드

```
require 'vendor/autoload.php';
```

코드 재사용(& 배포)

Let's face it: PHP has had a rocky history with package management, and as a result, it is pretty rare to find a developer who actively uses systems like PEAR. Instead, most developers have chosen their favorite framework, which has code specifically written for it to handle various things, like DB interaction, ORM's, OAuth, Amazon S3 integration, etc.

The downside here, though, is that switching frameworks (or returning to not using a framework at all) can be a nightmare, as it involves relearning everything to use brand new tools – and that is no easy task. Well, Composer can fix that!

Introduction

Composer sets out to solve this situation by positioning itself as "the glue between all projects" – meaning that packages can be written, developed and shared in a format that other developers can plug into other applications with ease.

> *"The glue between all projects."*

This article sets out to show you how to install and work with Composer packages. By the end of this article, you will be able to plug and play with chunks of code in any framework, whether you work with CodeIgniter, FuelPHP, Laravel, Symfony2, Lithium, Yii, Zend... or anything else.

Why & What

쓰면 뭐가 좋나(with 시연)

궁금한 거

라이브러리/패키지를 공유하는 데에 있어

왜 version control system을 쓰는 것 보다

Composer를 쓰는 게 더 나은 건가요?

Step 5 - Real World

Currently, most projects store all PHP dependencies in the main code repository; so, if you are using the Facebook SDK, for example, you just shove that version into your code by copy-pasting the code from GitHub or extracing a ZIP file. Then, you add it to your versioning system and push the changes.

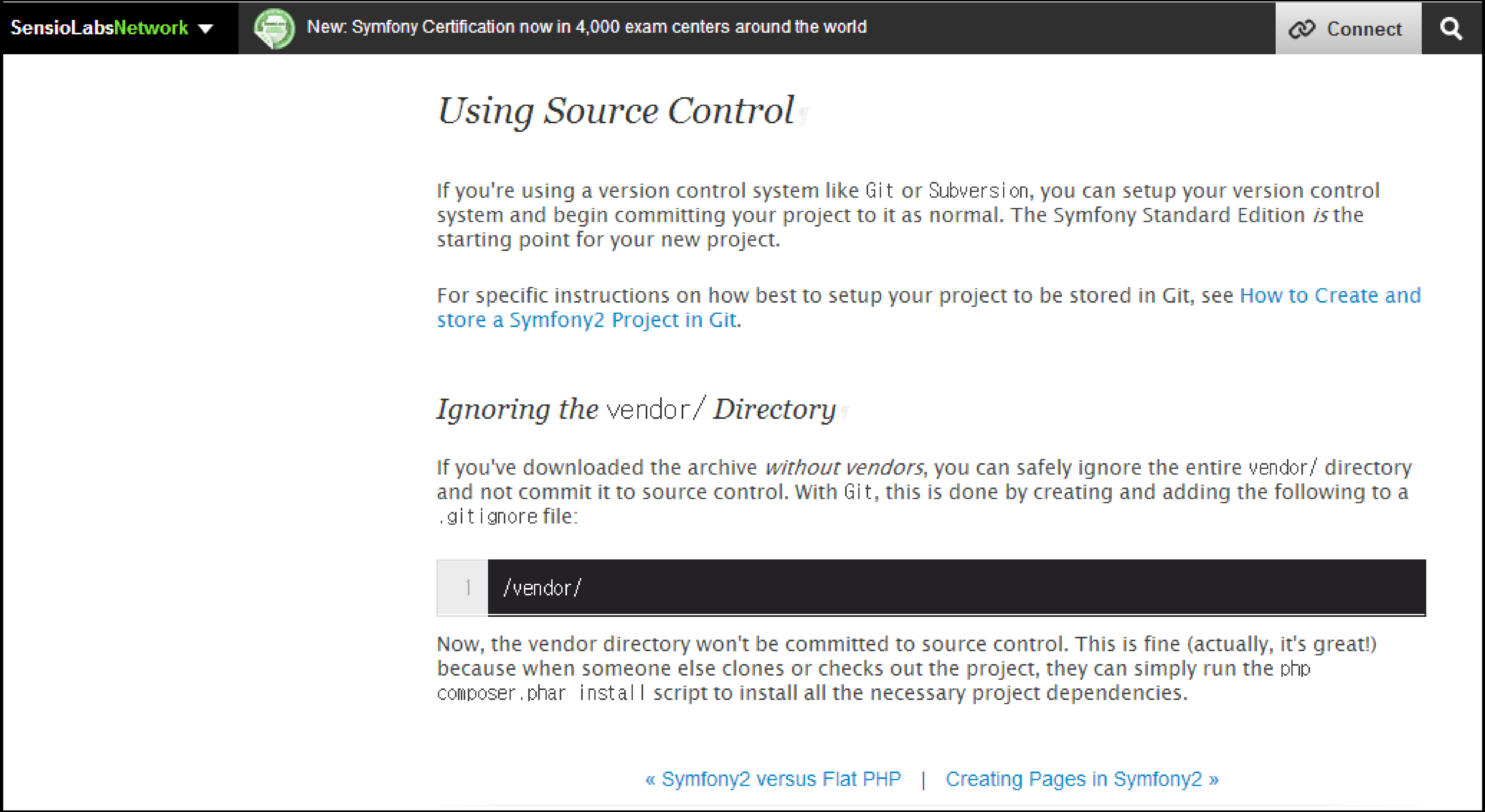
That version then sits with your code as a static file, which, at some point, you may or may not remember to upgrade – **IF** you notice that Facebook have released an updated version. The new version of the file goes over the top and you push those new changes, too.

> “*If you want to be really clever, you can automate the whole process.*”

You *can* use Composer to avoid needing to pay attention to the versions, and just run an update, and commit all the changes. But why have loads of code in your repository that you don't need to have in there?

The neatest solution is to add `vendors/` to your "Ignore" list (E.g: .gitignore) and keep your code out of there entirely. When you deploy code to your hosts, you can just run `composer install` or `composer update`.

If you want to be really clever, you can automate the whole process, so if you have hosting in the cloud, you can set up hooks to run `composer install` as soon as your new code is pushed!



SensioLabsNetwork ▾ New: Symfony Certification now in 4,000 exam centers around the world Connect

Using Source Control

If you're using a version control system like `Git` or `Subversion`, you can setup your version control system and begin committing your project to it as normal. The Symfony Standard Edition *is* the starting point for your new project.

For specific instructions on how best to setup your project to be stored in Git, see How to Create and store a Symfony2 Project in Git.

Ignoring the `vendor/` Directory

If you've downloaded the archive *without vendors*, you can safely ignore the entire `vendor/` directory and not commit it to source control. With `Git`, this is done by creating and adding the following to a `.gitignore` file:

```
/vendor/
```

Now, the vendor directory won't be committed to source control. This is fine (actually, it's great!) because when someone else clones or checks out the project, they can simply run the `php composer.phar install` script to install all the necessary project dependencies.

« Symfony2 versus Flat PHP | Creating Pages in Symfony2 »

http://symfony.com/doc/current/book/installation.html

선택 아닌 필수

끝