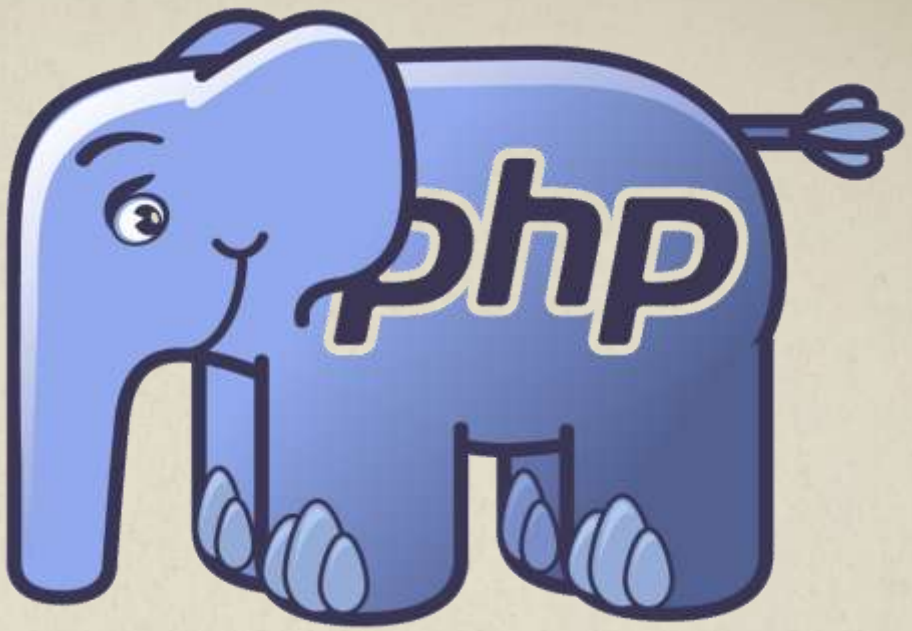



# PHP: ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

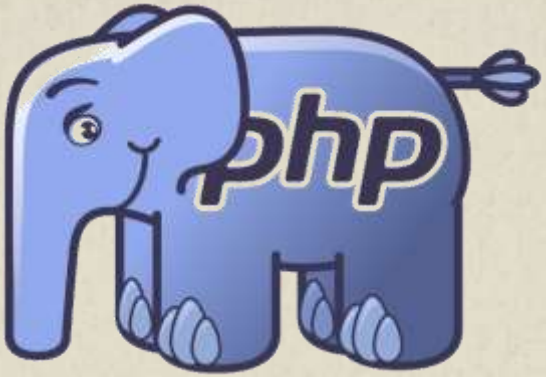
**УРОК 6.** СОВРЕМЕННЫЕ СТАНДАРТЫ PHP

# ОБРАЩЕНИЕ К «СКЛАДЧИКАМ»

Я считаю складчины – полной ерундой. Невозможно научиться чему-то, просматривая видео. Без домашних заданий, без общения с преподавателями и коллегами. Покупая в складчину видеозаписи курсов, вы вредите прежде всего самим себе, создавая иллюзию «обучения». И поддерживаете каких-то мутных личностей-«организаторов», имеющих свой процент.

Впрочем, дело ваше.

Однако, если вы хотите по-настоящему учиться – приходите. Адрес есть на слайдах. Напишите в поддержку, мол «я складчик, но я хочу учиться». Скидку гарантирую ☺




# ПРОСТРАНСТВА ИМЕН И АВТОЗАГРУЗКА

## Пространство имён – своеобразная «папка» в которой может находиться класс

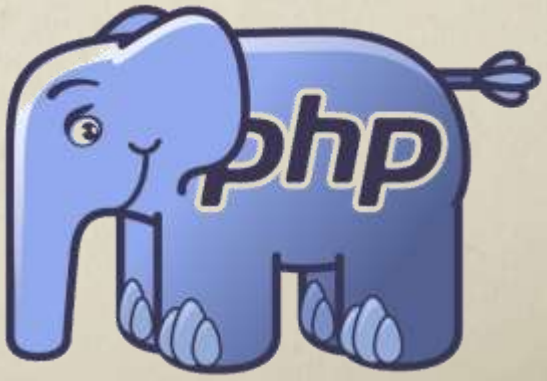
```
namespace App\Models;

class News {
  ...
}
```

- Конструкция `namespace` пишется в начале файла и действует до его конца.
- Полным именем класса в этом примере будет `\App\Models\News`

```
$model = new \App\Models\News();
```

- Можно использовать конструкцию `use`, чтобы задать короткие синонимы для «длинных» имен:
  ```
  use \App\Models\News as Model;
  $model = new Model();
  ```
- Считается, что классы, у которых не указано пространство имён, находятся в «корневом». То есть можно писать `\Config` или просто `Config`
- **ВАЖНО!** В функцию автозагрузки будет передано ПОЛНОЕ имя класса, включая пространство имён от корневого!

# NAMESPACE

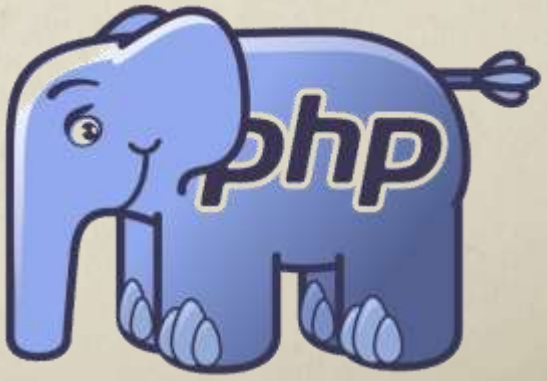


## PSR – это стандарты, обязательные для соблюдения всем программистами на PHP

- **PSR-1**
  **Основной стандарт кодирования**
  - Код, написанный для PHP 5.3 и более новых версий ОБЯЗАН использовать пространства имён
  - Имена классов и пространств имён обязаны быть написаны в стиле `StudlyCaps`
  - Первое пространство имён – это имя вендора, то есть «производителя» кода

```
Symfony\Core\Request

Zend\Mvc\Router

App\Controllers\Admin\News
```

PSR



**Задача: проект состоит из нескольких частей, каждая из них – со своей автозагрузкой.**

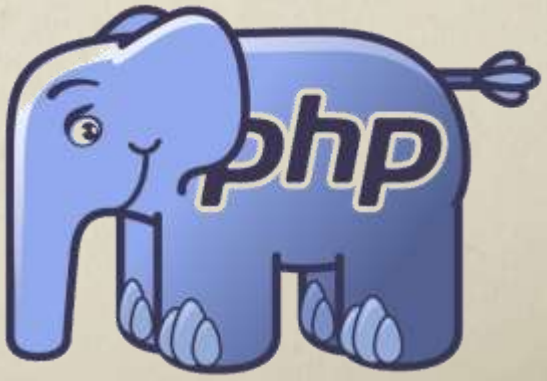
Задача решается так называемой «цепочкой автозагрузки».

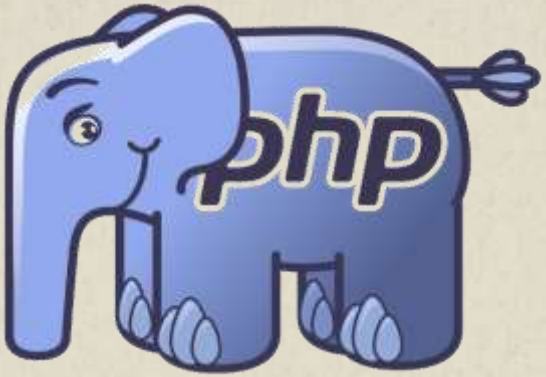
- `spl_autoload_register(callable $autoload)`
  Функция из состава SPL. Регистрирует заданную функцию в очереди в качестве реализации автозагрузки.
  Она создает очередь из функций автозагрузки в порядке их определения в скрипте.

```
function my_autoload($class) {
  ...
}

spl_autoload_register('my_autoload');

spl_autoload_register(function ($class) {
  ...
});
```

*NB. Если в вашем скрипте реализована функция __autoload(), ее необходимо явно зарегистрировать в очереди!*

## ЦЕПОЧКА АВТОЗАГРУЗКИ



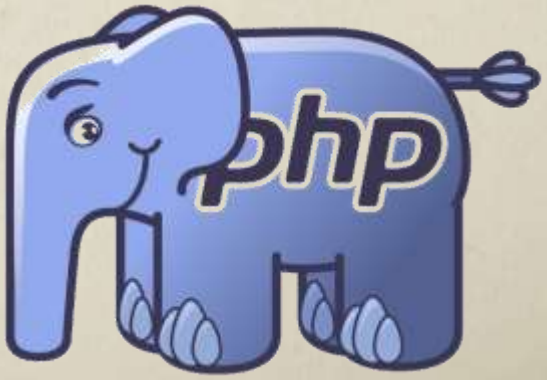



# ДРУГИЕ СТАНДАРТЫ PSR

# PSR-1
## Основной стандарт кодирования

- Используются только теги `<?php` и `<?=`
- Кодировка – только UTF-8
- Константы пишутся `В_ВЕРХНЕМ_РЕГИСТРЕ`
- Имена методов `пишутсяВотТак`
- Правило «побочных эффектов»:
  Каждый файл
  - ЛИБО определяет какие-либо сущности: классы, функции, константы
  - ЛИБО производит какие-либо активные действия


PSR-1

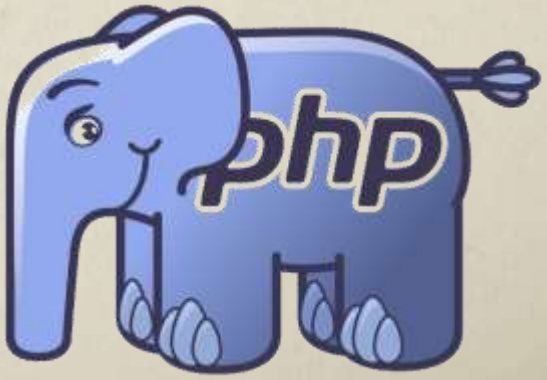


# PSR-2
## Руководство по стилю кода

- 4 пробела для табуляции
- Желательно не более 120 символов в строке
- Не жалейте переводов строк:
  - Открывающая скобка `{` класса на новой строке
  - Скобка `{` функции – на новой строке
  - Пустая строка после `namespace` и после `use`
- Видимость для методов обязательна
- `abstract` и `final` пишем до видимости
- `static` – после
- `{` для `if, while` и пр. – на той же строке!



PSR-2

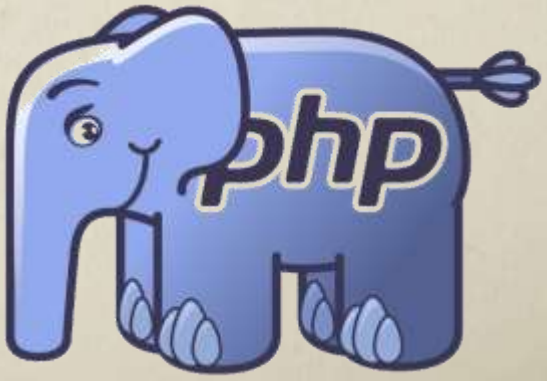


# PSR-3
## Интерфейс LoggerInterface

```
interface LoggerInterface
{
 public function
  emergency($message, array $context = []);

 public function
  alert($message, array $context = []);

 public function
  critical($message, array $context = []);

 public function
  error($message, array $context = []);

 public function
  warning($message, array $context = []);

 public function
  notice($message, array $context = []);

 public function
  info($message, array $context = []);

 public function
  debug($message, array $context = []);

 public function
  log($level, $message, array $context = []);
}
```



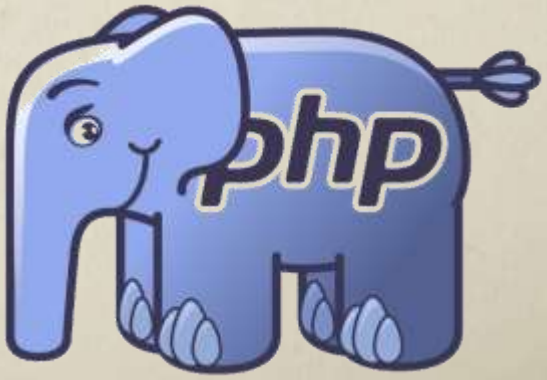
PSR-3

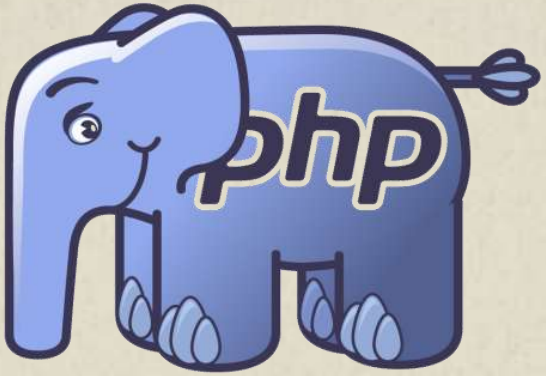


- **PSR-4**
  Стандарт автозагрузки, уже знакомый нам
- **PSR-5**
  ***Черновик***
  Стандарт на документацию PHPDoc
- **PSR-6**
  Интерфейс кэширования
- **PSR-7**
  Интерфейсы для протокола HTTP

И еще пара десятков стандартов, находящихся в черновиках…

## PSR






# COMPOSER

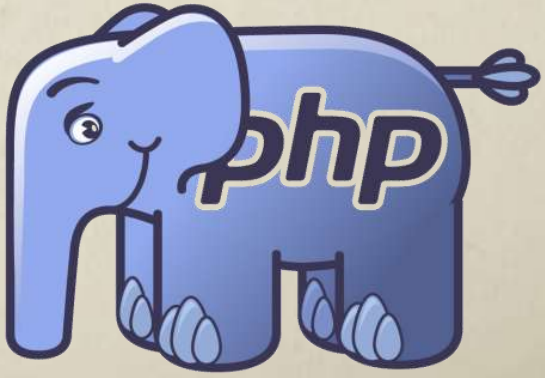
## Composer

Инструмент для управления зависимостями в приложениях на PHP.

1. Поместите в корень своего проекта файл `composer.json`
2. Опишите в нём зависимости – какие пакеты вы хотите использовать в своем проекте
3. Дайте команду `composer install`
4. Подключите к проекту файл `vendor/autoload.php`
5. **PROFIT!!!**


```
{
"require": {
    "pr-of-it/t4": "dev-master"
  }
}
```

## COMPOSER

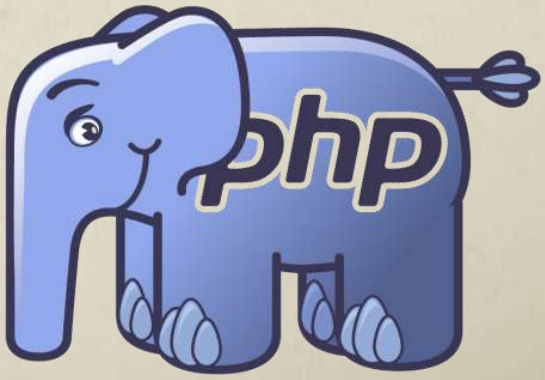


## Что важно знать о Composer?

- Есть каталог пакетов, называется Packagist: https://packagist.org/
- В нем собраны десятки тысяч пакетов, от библиотек в один класс и до больших фреймворков
- Вы можете легко оформить свой код, как пакет и выложить на Packagist
- Однако пакет можно включить в проект и без каталога – например напрямую с сервиса Github


- Важная часть composer – управление версиями пакетов. И команда `composer update`
- А еще он умный. И создает локальный кэш пакетов, а не качает их каждый раз!

COMPOSER



# ДО ВСТРЕЧИ НА СЛЕДУЮЩЕМ УРОКЕ!

**ВИДЕОЗАПИСЬ, СЛАЙДЫ, ПРЕЗЕНТАЦИЯ
И ДОМАШНЕЕ ЗАДАНИЕ
БУДУТ ВЫЛОЖЕНЫ ДО 10 УТРА СЛЕДУЮЩЕГО ДНЯ**

ProfIT