

Стандарты в РНР

Зачем нужны стандарты в РНР



Зачем нужны стандарты в PHP

Код без стандартов

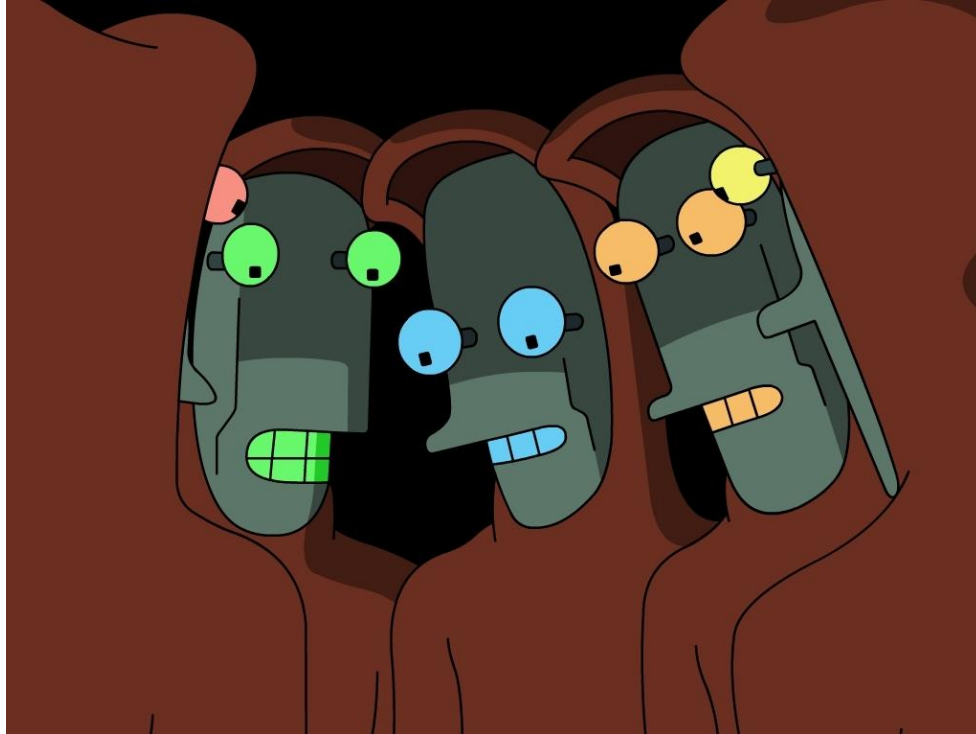
- Не содержит единого стиля
- Тяжело читаем
- Тяжело поддерживать
- Тратится много времени на разработку

Код который следует стандартам

- Следует единому стилю
- Легко читаем
- Минимальная стоимость внесения правок
- Легко оформить как библиотеку и выложить в Open Source
- Количество людей которые вас ненавидят стремиться нулю

PHP - FIG

(PHP Framework Interoperability Group)



Цели которые преследует PHP-FIG

Основная цель PHP-FIG это развитие экосистемы PHP и продвижение стандартов для написания хорошего кода.

Разработка и публикация стандартов.

Находить способ совместной работы нескольких фреймворков

Для разработки стандартов используется опыт других компаний и разработчиков, а также разработки и эксперименты самой инициативной группы.

PHP Standards Recommendations

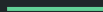
PSR

Устаревшие

Принятые

Черновики

Отклоненные



Стандарты PSR по категориям

Устаревшие

- PSR-0

Принятые

- PSR-1
- PSR-2
- PSR-3
- PSR-4
- PSR-6
- PSR-7
- PSR-11
- PSR-13
- PSR-15
- PSR-16
- PSR-17
- PSR-18

Черновики

- PSR-5
- PSR-12
- PSR-14
- PSR-19

Отклоненные

- PSR-8
- PSR-9
- PSR-10

Стандарты PSR

Номер PSR	Название	Статус
PSR-0	Стандарт автозагрузки	Устаревший
PSR-1	Основной стандарт кодирования	Принятый
PSR-2	Рекомендация по оформлению кода	Принятый
PSR-3	Интерфейс протоколирования	Принятый
PSR-4	Улучшенный стандарт автозагрузки	Принятый
PSR-6	Интерфейс кеширования	Принятый
PSR-7	Интерфейс HTTP сообщений	Принятый

Стандарты PSR

Номер PSR	Название	Статус
PSR-11	Интерфейс контейнеров	Принятый
PSR-13	Стандарт ссылок гипермедиа	Принятый
PSR-15	Стандарт HTTP обработчиков	Принятый
PSR-16	Простой кеш	Принятый
PSR-17	Стандарт HTTP фабрик	Принятый
PSR-18	Стандарт HTTP клиента	Принятый

А теперь немного детальней ...



Стандарт автозагрузки: PSR-0

Общие положения:

- Пространство имен и класс должны иметь следующую структуру:

`|<Vendor Name>|(<Namespace>|)*<Class Name>`

- Каждое пространство имен должно содержать пространство верхнего уровня («Vendor Name»).
- Каждое пространство имен может иметь сколько угодно уровней.

Стандарт автозагрузки: PSR-0

Общие положения:

- Каждый разделитель пространства имен конвертируется в DIRECTORY_SEPARATOR при загрузке.
- Каждый символ «_» в CLASS NAME конвертируется в DIRECTORY_SEPARATOR.
- К полностью определенному пространству имен и классу добавляется «.php» при загрузке.

Стандарт автозагрузки: PSR-0

Примеры:

- `\Doctrine\Common\IsolatedClassLoader =>`
`/path/to/project/lib/vendor/Doctrine/Common/IsolatedClassLoader.php`
- `\Symfony\Core\Request =>`
`/path/to/project/lib/vendor/Symfony/Core/Request.php`
- `\Zend\Acl =>` `/path/to/project/lib/vendor/Zend/Acl.php`
- `\Zend\Mail\Message =>` `/path/to/project/lib/vendor/Zend/Mail/Message.php`

Стандарт автозагрузки: PSR-0

Примеры с “_” в качестве разделителя:

- `\namespace\package\Class_Name =>`
`/path/to/project/lib/vendor/namespace/package/Class/Name.php`
- `\namespace\package_name\Class_Name =>`
`/path/to/project/lib/vendor/namespace/package_name/Class/Name.php`

Стандарт автозагрузки: PSR-0

```
<?php

function autoload($className)
{
    $className = ltrim($className, '\\');
    $fileName  = '';
    $namespace = '';
    if ($lastNsPos = strrpos($className, '\\')) {
        $namespace = substr($className, 0, $lastNsPos);
        $className = substr($className, $lastNsPos + 1);
        $fileName  = str_replace('\\', DIRECTORY_SEPARATOR, $namespace) . DIRECTORY_SEPARATOR;
    }
    $fileName .= str_replace('_', DIRECTORY_SEPARATOR, $className) . '.php';

    require $fileName;
}

spl_autoload_register('autoload');
```

Основные стандарты кода: PSR-1

Общие положения:

- В файлах должны использоваться только теги `<?php` и `<?.`
- В файлах следует либо объявлять структуры (классы, функции, константы и т.п.), либо генерировать побочные эффекты (выполнять действия).
- В файлах должна использоваться только кодировка UTF-8 without BOM.
- Имена пространств и классы должны следовать PSR-0.
- Имена классов должны быть объявлены в нотации StudlyCaps.
- Константы класса должны быть объявлены в верхнем регистре, разделенные подчеркиваниями.
- Методы должны быть объявлены в нотации camelCase.

Основные стандарты кода: PSR-1

Что такое “побочные эффекты”

Под “побочными эффектами” понимается реализация логики, не связанной с объявлением классов, функций, констант и т.п. – даже подключение внешнего файла уже является “побочным эффектом”.

“Побочные эффекты” включают (но не ограничиваются этим перечнем): передачу данных в выходной поток, явное использование `require` или `include`, изменение настроек, генерирование ошибочных ситуаций или порождение исключений, изменение глобальных или локальных переменных, чтение из файла или запись в файл и т.п.

Основные стандарты кода: PSR-1

```
<?php
// побочный эффект: изменение настроек
ini_set('error_reporting', E_ALL);
// побочный эффект: подключение файла
include "file.php";
// побочный эффект: передача данных в выходной поток
echo "\n";
// объявление
function foo()
{
    // тело функции
}
```

Основные стандарты кода: PSR-1

```
<?php
// объявление
function foo()
{
    // тело функции
}
// условное объявление -- это НЕ побочный эффект
if (! function_exists('bar')) {
    function bar()
    {
        // тело функции
    }
}
```

Рекомендации по оформлению кода: PSR-2

Общие положения:

- Код должен быть оформлен согласно стандарту PSR-1.
- Для оформления отступов должны использоваться четыре пробела (но не знак табуляции).
- Недопустимо жёстко ограничивать длину строки; мягкое ограничение должно составлять 120 символов; следует стараться, чтобы длина строки составляла 80 символов или менее.
- После определения пространства имён (namespace) и после блока импорта пространств имён (use) должна быть одна пустая строка.

Рекомендации по оформлению кода: PSR-2

Общие положения:

- Открывающая фигурная скобка в определении класса должна располагаться на новой строке, а закрывающая фигурная скобка должна располагаться на следующей строке после тела класса.
- Открывающая фигурная скобка в определении метода должна располагаться на новой строке, а закрывающая фигурная скобка должна располагаться на следующей строке после тела метода.
- Область видимости должна быть указана явно для всех свойств и методов; модификаторы `abstract` и `final` должны располагаться перед модификаторами области видимости; модификатор `static` должен располагаться после модификаторов области видимости.

Рекомендации по оформлению кода: PSR-2

Общие положения:

- После ключевых слов в управляющих конструкциях должен располагаться один пробел, а после вызовов функций и методов – не должен.
- Открывающая фигурная скобка в управляющих конструкциях должна располагаться в той же строке, что и сама конструкция, а закрывающая фигурная скобка должна располагаться на следующей строке после тела конструкции.
- После открывающей круглой скобки и перед закрывающей круглой скобкой в управляющих конструкциях не должно быть пробела.

```
<?php
namespace Vendor\Package;

use FooInterface;
use BarClass as Bar;
use OtherVendor\OtherPackage\BazClass;

class Foo extends Bar implements FooInterface
{
    final public static function sampleFunction($a, $b = null)
    {
        if ($a === $b) {
            bar();
        } elseif ($a > $b) {
            $foo->bar($arg1);
        } else {
            BazClass::bar($arg2, $arg3);
        }
    }
}
```

Рекомендации по оформлению кода: PSR-2

Требование к файлам

- Во всех файлах с PHP-кодом должен быть использован Unix-вариант переноса строк (Unix line feed, т.е. \n).
- В конце каждого файла с PHP-кодом должна быть одна пустая строка. Закрывающий тег ?> должен отсутствовать в файлах, содержащих только PHP-код. [пример](#)

Рекомендации по оформлению кода: PSR-2

Требование к строкам

- Не должно быть жесткого ограничения длины строки.
- Мягкое ограничение длины строки должно составлять 120 символов; автоматические системы проверки стиля должны выдавать предупреждение при превышении этого ограничения, но не должны считать это ошибочной ситуацией.
- Следует стараться, чтобы длина строки составляла 80 символов или менее; более длинные строки следует разбивать на несколько отдельных строк, длина каждой из которых не превышала бы 80 символов. [пример](#)

Рекомендации по оформлению кода: PSR-2

Требование к строкам

- В конце непустых строк не должно быть пробелов.
- Пустые строки могут быть добавлены в код для повышения удобочитаемости и разделения блоков кода.
- В одной строке не должно быть более одного выражения. [пример](#)

Рекомендации по оформлению кода: PSR-2

Требование к отступам

- Для оформления отступов должны использоваться четыре пробела (но не знак табуляции).

Примечание: использование только лишь пробелов (без смешивания их с табуляциями) позволяет избежать проблем с обработкой истории изменения кода, определением самих изменений, патчами и комментариями.

Использование пробелов также позволяет легко добавлять небольшие отступы для выравнивания отдельных вложенных строк.

Рекомендации по оформлению кода: PSR-2

Требование к ключевым слова и константам true, false, null

- Ключевые слова PHP должны быть написаны в нижнем регистре.
- Константы PHP true, false и null должны быть написаны в нижнем регистре.

[пример](#)

Рекомендации по оформлению кода: PSR-2

Требование к пространству имен и импорту

- В случае наличия определения пространства имён, после него должна располагаться одна пустая строка.
- В случае наличия импорта пространств имён, он должен располагаться после определения пространства имен.
- При реализации импорта каждое пространство имен должно импортироваться отдельно (со своим ключевым словом `use`).
- После блока импорта должна быть одна пустая строка. [пример](#)

[Другие примеры](#)

Улучшенный стандарт автозагрузки: PSR-4

Общие положения:

Данный стандарт описывает спецификацию автозагрузки классов на основе путей к файлам. Он полностью совместим (и может использоваться как дополнение) с любой другой спецификацией автозагрузки, включая PSR-0. Данный стандарт также описывает правила размещения файлов, предназначенных для автозагрузки.

В реализации автозагрузчика недопустимо порождать исключения, ошибочные ситуации любого уровня и не следует возвращать какое бы то ни было значение.

Улучшенный стандарт автозагрузки: PSR-4

Общие положения:

- Полностью определенное имя класса должно начинаться с пространства имён высшего уровня, указывающего на разработчика кода (“имя производителя”).
- Полностью определённое имя класса может включать в себя одно или более подпространств имен.
- Полностью определенное имя класса должно заканчиваться именем класса.
- Символ `_` (“знак подчеркивания”) не обладает никаким особенным значением в полностью определённом имени класса.

Улучшенный стандарт автозагрузки: PSR-4

Абсолютное имя класса	Префикс пространства имён	Базовая директория	Путь к файлу
\Acme\Log\Writer\File_Writer	Acme\Log\Writer	./acme-log-writer/lib/	./acme-log-writer/lib/File_Writer.php
\Aura\Web\Response>Status	Aura\Web	/path/to/aura-web/src/	/path/to/aura-web/src/Response/Status.php
\Symfony\Core\Request	Symfony\Core	./vendor/Symfony/Core/ /	./vendor/Symfony/Core/Request.php
\Zend\Acl	Zend	/usr/includes/Zend/	/usr/includes/Zend/Acl.php

Спасибо за внимание

email: oleg.grynko@aengizer.dp.ua

<https://github.com/php-fig/fig-standards>

```
<?php
```

[Назад](#)

```
class Example
```

```
{
```

```
    // какой то код...
```

```
}
```

```
//^ выше одна пустая строка
```

```
<?php
```

```
class Example
```

```
{
```

```
    public function thisIsALongMethodName(string $methodArgument, int $anotherArgument)
```

```
    {
```

```
        // какой то код
```

```
    }
```

```
}
```

```
<?php
```

```
// Пример с переносом аргументов
```

```
class Example
```

```
{
```

```
    public function thisIsALongMethodName(
```

```
        string $methodArgument,
```

```
        int $anotherArgument
```

```
    ) {
```

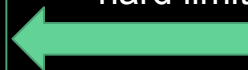
```
        // какой то код
```

```
    }
```

```
}
```

[Назад](#)

hard limit 80



```
<?php

class Example
{
    private $users = [];

    private function doSomeCoolStuff()
    {
        // Не надо писать все в одну строку
        $name = 'Vasia'; $this->users[] = $name;
        if (!empty($this->users)) {
            // И тут тоже
            $this->doSomeAnotherCheck; return $this->users;
        }
    }
}
```

Хороший пример

```
<?php
```

[Назад](#)

```
class Example
```

```
{
```

```
    private $users = [];
```

```
    private function doSomeCoolStuff()
```

```
    {
```

```
        // Так то лучше
```

```
        $name = 'Vasia';
```

```
        $this->users[] = $name;
```

```
        if (!empty($this->users)) {
```

```
            // И тут тоже лучше
```

```
            $this->doSomeAnotherCheck;
```

```
            return $this->users;
```

```
        }
```

```
    }
```

```
}
```

Можете использовать пустую строку для разделения

// Так хорошо

```
function foo($name = null)
```

```
{
```

```
    $isChecked = true;
```

```
    if ($name != null) {
```

```
        return $name;
```

```
    }
```

```
    return false;
```

```
}
```

// А так очень плохо

```
function baz($name = NULL)
```

```
{
```

```
    $isChecked = TRUE;
```

```
    IF ($name != null) {
```

```
        RETURN $name;
```

```
    }
```

```
    RETURN FALSE;
```

```
}
```

Назад

```
<?php
```

```
namespace Vendor\Package;
```

```
use FooClass;
```

```
use BarClass as Bar;
```

```
use OtherVendor\OtherPackage\BazClass;
```

```
class Example
```

```
{
```

```
    // какой то код
```

```
}
```

[Назад](#)

```
<?php
```

```
namespace Vendor\Package;
```

```
use FooClass;
```

```
use BarClass as Bar;
```

```
use OtherVendor\OtherPackage\BazClass;
```

extends и implements должны
быть на одной строке

```
class ClassName extends ParentClass implements \ArrayAccess, \Countable
```

```
{
```

```
    // constants, properties, methods
```

```
}
```

Открывающая скобка
должны быть на новой
строке, закрывающая на
новой строке после тела

[Назад](#)


```
<?php
```

```
namespace Vendor\Package;
```

```
use FooClass;
```

```
use BarClass as Bar;
```

```
use OtherVendor\OtherPackage\BazClass;
```

```
class ClassName extends ParentClass implements
```

```
    \ArrayAccess,
```

```
    \Countable,
```

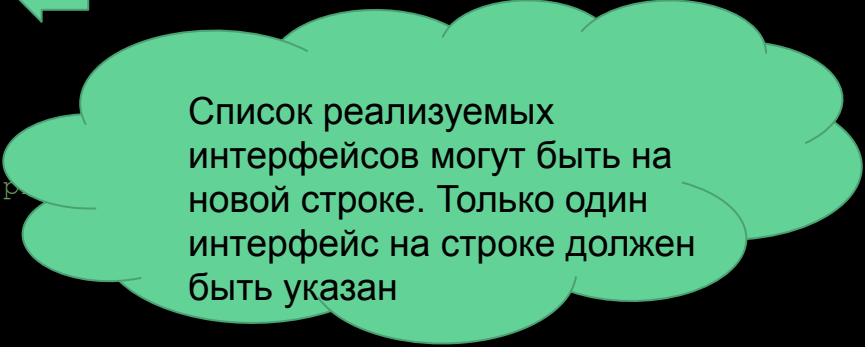
```
    \Serializable
```

```
{
```

```
    // constants, p
```

```
}
```

[Назад](#)



Список реализуемых интерфейсов могут быть на новой строке. Только один интерфейс на строке должен быть указан

```
<?php
```

```
namespace Vendor\Package;
```

```
class ClassName
```

```
{  
    Никаких var здесь  
    public $foo = null;  
}
```



Области видимости должны
быть указаны явно. Не
используйте _ для
приватных и защищенных
свойств и методов.

[Назад](#)

```
<?php
```

```
namespace Vendor\Package;
```

```
class ClassName
```

```
{
```

```
    public $foo = null;
```

```
    public function doSomeStuff()
```

```
{
```

```
}
```

```
}
```

[Назад](#)



После имени метода не
должно быть пробела.

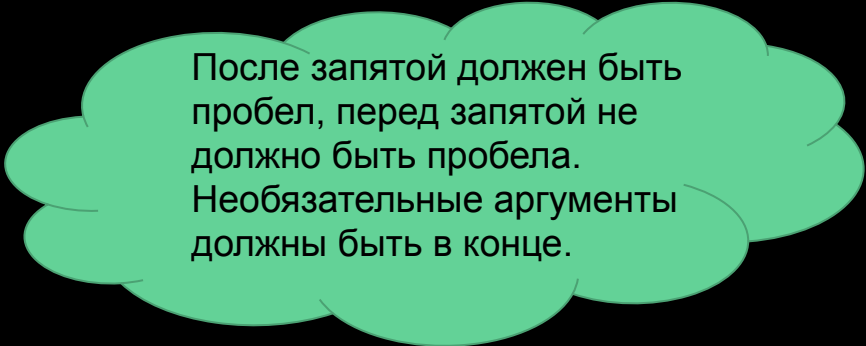
Скобки должны быть на
новой строке после метода

```
<?php
```

```
namespace Vendor\Package;
```

```
class ClassName
```

```
{  
    public function foo($arg1, &$arg2, $arg3 = [])  
    {  
        // method body  
    }  
}
```



После запятой должен быть пробел, перед запятой не должно быть пробела.
Необязательные аргументы должны быть в конце.

[Назад](#)

```
<?php
```

```
namespace Vendor\Package;
```



```
abstract class ClassName
```

```
{
```



```
    protected static $foo;
```

```
    abstract protected function zim();
```

```
➡ final public static function bar()
```

```
{
```

```
    // method body
```

```
}
```

```
}
```

Назад

Ключевые слова `abstract` и `final`, в случае их наличия, должны располагаться перед указанием области видимости.

Ключевое слово `static`, в случае его наличия, должно располагаться после указания области видимости.

```
<?php
bar();
$foo->bar($arg1);
Foo::bar($arg2, $arg3);
```



```
<?php
$foo->bar(
    $longArgument,
    $longerArgument,
    $muchLongerArgument
);
```



[Назад](#)

В коде вызова функций и методов не должно быть пробела между именем функции или метода и открывающей круглой скобкой, не должно быть пробела после открывающей круглой скобки, не должно быть пробела перед закрывающей круглой скобкой. В списке аргументов не должно быть пробелов перед запятыми, но должен быть пробел после каждой запятой.

```
<?php
if ($expr1) {
    // if body
} elseif ($expr2) {
    //elseif body
} else {
    // else body;
}
```

[Назад](#)

После ключевого слова, определяющего управляющую конструкцию, должен быть один пробел.

После открывающих круглых скобок не должно быть пробелов.

Перед закрывающими круглыми скобками не должно быть пробелов
Между закрывающей круглой скобкой и открывающей фигурной скобкой должен быть один пробел.

```
<?php
switch ($expr) {
    case 0:
        echo 'First case, with a break';
        break;
    case 1:
        echo 'Second case, which falls through';
        // no break
    case 2:
    case 3:
    case 4:
        echo 'Third case, return instead of break';
        return;
    default:
        echo 'Default case';
        break;
}
```

[Назад](#)

Выражение case должно быть смещено на один отступ (четыре пробела) от switch, а ключевое слово break (или иное слово, обозначающее выход из конструкции) должно располагаться на том же уровне отступов, что и тело case.

В том случае, когда в непустом теле case умышленно не используется break, должен быть комментарий в стиле // no break.


```
<?php
while ($expr) {
    // structure body
}
```

```
do {
    // structure body;
} while ($expr);
```

```
for ($i = 0; $i < 10; $i++) {
    // for body
}
```

```
foreach ($iterable as $key => $value) {
    // foreach body
}
```

```
try {
    // try body
} catch (FirstExceptionType $e) {
    // catch body
}
```

[Назад](#)

Примеры оформления циклов и try catch.