

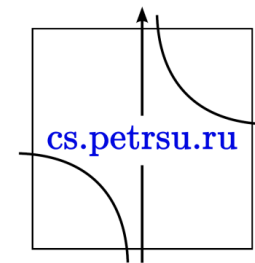
Разработка и анализ технической документации

Системы автоматической генерации документации

Глава №3

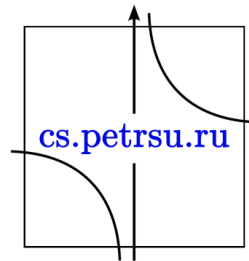
Что это такое

- Генератор документации — программа или пакет программ, позволяющая получать документацию, предназначенную для программистов (документация на API) и/или для конечных пользователей системы, по особым образом комментированному исходному коду и, в некоторых случаях, по исполняемым модулям (полученным на выходе компилятора).



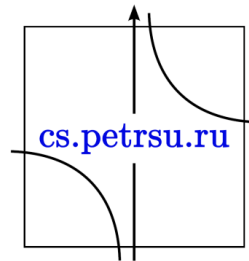
Документирующий комментарий — это особым образом оформленный комментарий к объекту программы, предназначенный для использования каким-либо конкретным генератором документации.

- Содержит информацию об авторе кода, описывается назначение объекта программы, смысл входных и выходных параметров — для функции/процедуры и т.д.
- Как правило, оформляются как многострочные комментарии в стиле языка Си.
- Комментарий должен находиться перед документируемым элементом.
- Первым символом в комментарии (и в начале строк комментария) должен быть *.
- Блоки разделяются пустыми строками.



Общий вид

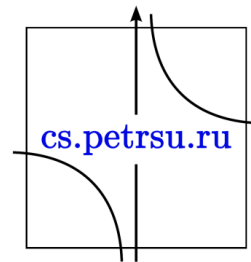
```
/**  
 * Имя или краткое описание объекта  
 *  
 * Развернутое описание  
 *  
 * @имя_дескриптора значение  
 * @return тип_данных  
 */
```



Тег, аннотация, команда, дескриптор

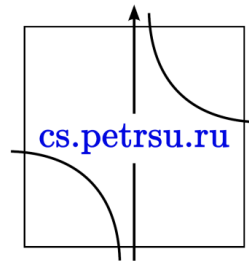
1. @имя

2. \имя



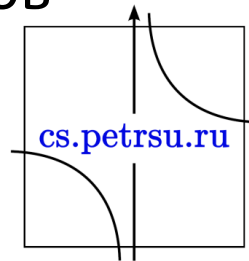
Doxygen

- кроссплатформенная система документирования исходников, которая поддерживает языки: C/C++, Obj-C, Python, Java, IDL, PHP, C#, Fortran, TCL и VHDL.
 - Несколько формов:
 - HTML (генерация онлайн документация для WEB)
 - XML
 - RTF
 - MAN
 - LATEX



Doxygen

- Документация извлекается непосредственно из исходного кода, что в дальнейшем значительно упрощает сопровождение, хранение и обновление документации проекта.
- Doxygen подразумевает внедрение специального синтаксиса, который используется для оформления блоков с комментариями.
- Синтаксис Doxygen следует применять для формирования шапок файлов, написания заголовков функций и методов, внутренних комментариев и прочего.



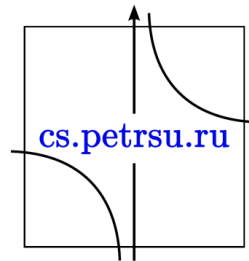
Правила

- Для многострочного комментария используется конструкция:

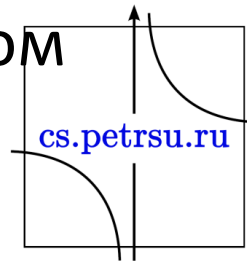
```
/** ваш комментарий */
```

- Для однострочного комментария используется конструкция:

```
/// ваш комментарий
```

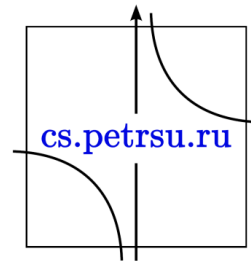


- Вывод Doxygen может содержать в себе таблицы, списки, ссылки, диаграммы, графы, иерархические указатели и т.п.
- Doxygen может помочь вам визуализировать отношения между различными элементами программы, путем отображения графических зависимостей, диаграмм наследования и взаимодействия, которые создаются автоматически.
- Можете настроить работу генератора Doxygen на извлечение структуры кода из недокументированных исходников. Полезно для быстрого поиска нужной информации в большом распределенном проекте.
- Doxygen понимает Markdown в комментариях.

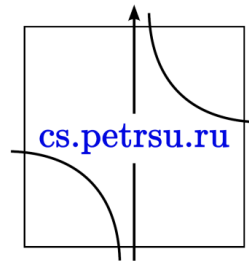


Документирование функций и методов

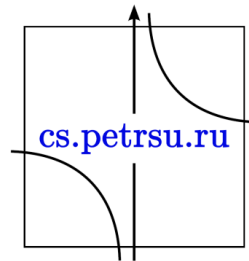
- **@arg**
Описание параметров, передаваемых функции.
- **@brief**
Краткое описание сущности.
- **@detailed**
Подробное описание сущности.
- **@example**
Указывает, что блок комментариев содержит пример использования чего-либо.



- **@internal**
Указывает, что фрагмент предназначен только для внутреннего использования
- **@param**
Описание параметров, передаваемых функции.
- **@pre**
Начальные условия
- **@post**
Конечные условия

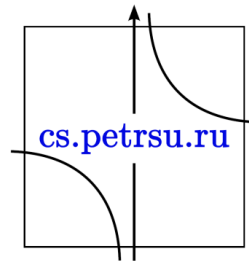


- **@return**
Описание значения, возвращаемого функцией.
- **@retval**
Для более подробного описания возвращаемых значений
- **@see**
Ссылки на классы, функции, методы, переменные, файлы, URL и пр.
- **@code - @endcode**
Для приведения примера работы

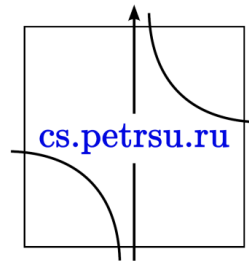


Команды документирования файлов

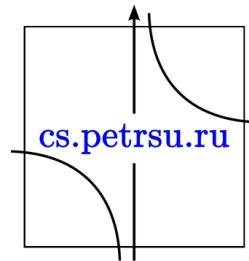
- **@mainpage**
Используется для указания основной (начальной) страницы проекта
Пример: @mainpage Главная страница
- **@page**
Служит для указания дополнительного раздела документации
Пример: @page page1 — Страница 1
- **@ref**
Используется для указания ссылок на другие страницы документации
Пример: @ref page1
- **@file**
Краткое описание файла
Пример: @file file.h



- **@section, @subsection**
Задание разделов и подразделов документации
- **@version**
Используется для указания версии
- **@author**
Служит для указания автора
Пример: @author Ivanov I.I.
- **@date**
Дата написания файла
Пример: @date February, 2021

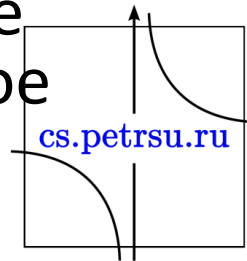


- **@link**
Ссылка на файл, код, описание, объект, документацию, email и пр.
Пример: @link test@example.com
- **@defgroup**
Задаёт группу в большом проекте для разделения на подпроекты.
Пример: @defgroup group Модуль авторизации
- **@ingroup, @addtogroup, @weakgroup**
Добавить объект в группу



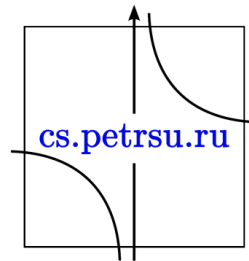
Ремарки в Doxygen

- Для добавления замечаний в код используются следующие команды/теги:
@attention
@bug
@todo
@note
@warning
- Имя команды определяет тип замечания/ремарки, и каждая команда вставляет соответствующую заметку. Для них Doxygen формирует отдельные списки, по которым можно быстро найти нужное место в коде.



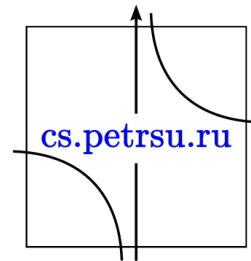
Пример использования ремарок

```
/// @warning Не менять значение этой переменной  
/// @todo Переписать функцию под новую архитектуру  
/// @bug Метод некорректно работает при  $a = 0$   
/// @note Подробнее читать в журнале тестирования
```



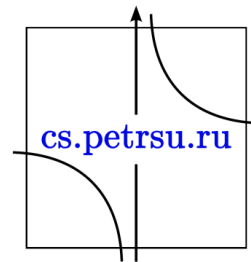
Другие команды

- **@class** :
Описание класса
- **@struct** :
Документирует структуру
- **@union** :
Документирует объединение
- **@enum** :
Документирует перечисление



Другие команды

- **@fn :**
Документирует функцию
- **@var :**
Документирует переменную
- **@def :**
Документирует макрос
- **@namespace:**
Документирует пространство имен

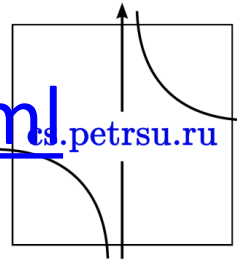


Другие команды

- **@typedef** :
Документирование объявления типа
- **@interface**:
Документирование интерфейса
- **@package** :
Документирование пакета

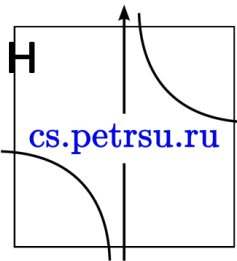
- И многие другие ...

<https://www.doxygen.nl/manual/index.html>

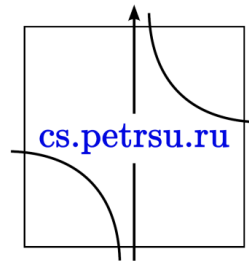


Настройка Doxygen

- Doxygen имеет множество тонких, управляемых пользователем настроек (далее тегов), хранящихся в обязательном конфигурационном файле (обычный текстовой файл, предназначенный для редактирования).
- Теги, хранящиеся в конфигурационном файле, описывают внешний вид документации, какие сущности и отношения между ними следует в неё включать, имя проекта, путь к анализируемым файлам и т.п.
- Конфигурационный файл достаточно настроить один раз под свои потребности и затем в новых проектах изменять лишь один тег – PROJECT_NAME.

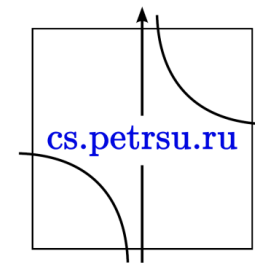


- Чтобы облегчить работу можно выполнить команду: `doxygen -g` и конфигурационный файл с именем `Doxyfile` будет создан автоматически.
- `Doxyfile` содержит настройки по умолчанию. Вы можете изменить их под свои нужды. Это сделать довольно просто благодаря содержащимся в `Doxyfile` пояснениям.



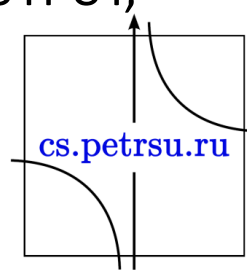
PROJECT_NAME	Имя проекта
QUIET	Режим тишины. Допустимые значения YES и NO (default: NO) YES, чтобы важные предупреждения не потерялись среди обилия информации
WARN_NO_PARAMDOC	YES - для вывода предупреждений о недокументированных аргументах функции
INPUT	Путь к анализируемому коду. Рекомендуется в корне проекта создать директорию docs содержащую Doxyfile с настройками, в таком случае значение этого параметра будет ../
FILE_PATTERNS	Шаблон для имени анализируемых файлов. Поскольку документироваться должны только открытые интерфейсы, то очевидно, что следует анализировать только заголовочные файлы: *.h
RECURSIVE	YES - рекурсивно анализировать файлы в поддиректориях (default: NO)
EXCLUDE_PATTERNS	Исключение из анализа директорий соответствующих маске. Пример: EXCLUDE_PATTERNS = */.svn/* */build/* */tests/* */sources/*
HAVE_DOT	YES - если установлен Graphviz и требуется визуализация связей.

DOXYFILE_ENCODING	Кодировка, которая используется для всех символов в данном файле настроек
OUTPUT_LANGUAGE	Устанавливает язык, на котором будет сгенерирована документация
PROJECT_NUMBER	Данный тэг может быть использован для указания номера проекта или его версии
PROJECT_BRIEF	Краткое однострочное описание проекта, которое размещается сверху каждой страницы и даёт общее представление о назначении проекта
OUTPUT_DIRECTORY	Абсолютный или относительный путь, по которому будет сгенерирована документация
...	
...	
...	



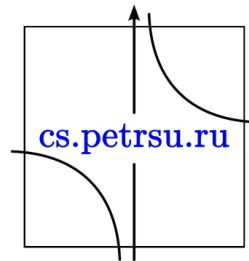
Запуск Doxygen

- Запуск системы автодокументирования прост – для этого надо запустить Doxygen указав ему путь к конфигурационному файлу:
`doxygen path/to/config-file`
- В зависимости от настроек будут созданы html, rtf, latex, xml, man каталоги в директории выгрузки, содержащие сгенерированную документацию в HTML, RTF, LaTeX, XML и Unix-Man Page форматах.
- По умолчанию, директория выгрузки - это каталог из которого запущен Doxygen. Путь к директории выгрузки, может быть изменен в конфигурационном файле с помощью тега `OUTPUT_DIRECTORY`
- Формат конкретного каталога с документацией в директории выгрузки может быть выбран с использованием тегов: `HTML_OUTPUT`, `RTF_OUTPUT`, `LATEX_OUTPUT`, `XML_OUTPUT` и `MAN_OUTPUT`
- Если `OUTPUT_DIRECTORY` не пуст, а такой директории выгрузки не существует, Doxygen создаст её для Вас.



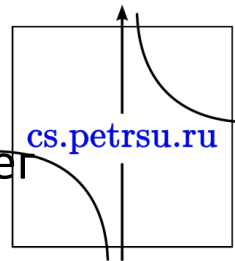
Вывод в HTML

- Для того, чтобы просмотреть сгенерированную HTML документацию нужно указать в адресной строке браузера путь к файлу `index.html` в каталоге `html`. Для получения лучших результатов, браузер должен поддерживать CSS.
- Некоторые из возможностей, такие как `GENERATE_TREEVIEW`, требуют браузер, который поддерживает DHTML и Javascript.
- Если Вы планируете использовать средства поиска (см. `SEARCHENGINE`), то просматривайте HTML вывод через WEB-сервер поддерживающий PHP (пример, Apache с предустановленным PHP модулем).



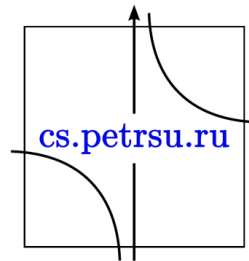
Вывод в LaTeX

- Для получения LaTeX документации необходимо скомпилировать сгенерированную документацию компилятором LaTeX. Для упрощения процесса компилирования Doxygen создает Makefile (содержание и цели зависят от значения тега USE_PDFLATEX).
- Если тег USE_PDFLATEX установлен в NO), то выполнив make без аргументов в каталоге latex будет сгенерирован dvi файл, вызывающий refman.dvi.
- Для генерации документации в формате PDF выполните make pdf (или make pdf_2on1). Будет построен файл refman.pdf.
- Если тег USE_PDFLATEX установлен в YES), то для конвертирования вывода в формат PDF достаточно запустить make без аргументов или make pdf_2on1.
- Чтобы получить более наглядный PDF файл, можно установить тег PDF_HYPERLINKS .



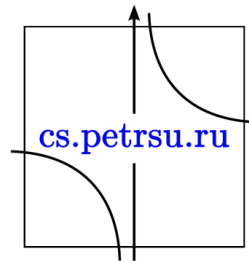
Doxyfile

- Посмотреть пример в терминале на сервере карра



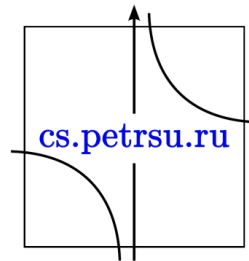
Примеры документации

- <https://api.kde.org/index.html>
- <https://vtk.org/doc/nightly/html/index.html>
- <http://www.abisource.com/doxygen/>



Markdown

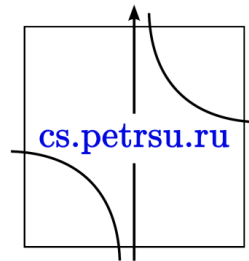
- Облегченный язык разметки, который является инструментом преобразования кода в HTML.
- Главной особенностью данного языка является максимально простой синтаксис, который служит для упрощения написания и чтения кода разметки, что, в свою очередь, позволяет легко его корректировать.
- Синтаксис Markdown достаточно ограничен, и соответствует лишь небольшому подмножеству элементов HTML.



Блочные элементы

Параграфы и разрывы строк

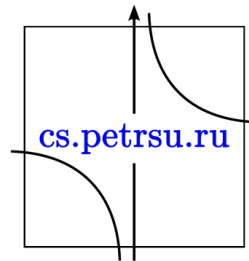
- Для того, чтобы создать параграф с использованием синтаксиса языка Markdown, достаточно отделить строки текста одной (или более) пустой строкой (пустой считается всякая строка, которая не содержит в себе ничего, кроме пробелов и символов табуляции). Для того, чтобы вставить видимый перенос строки (элемент `
`) необходимо окончить строку двумя пробелами и нажатием клавиши «Enter». Многие элементы синтаксиса Markdown выглядят и работают гораздо лучше в случае, когда их форматируют с помощью «жесткого перевода строк» (разделение строк, осуществленное самим пользователем, а не программой автоматически). К таким элементам относятся цитаты, списки и пр.



Блочные элементы

Заголовки

- Язык разметки Markdown поддерживает 2 стиля обозначения заголовков: подчеркивание и выделение символом («#»). Выделение заголовков с помощью подчеркивания производится знаками равенства («=») в случае, если заголовок первого уровня, и дефисами («-») в случае, если заголовок второго уровня. Количество знаков подчеркивания не ограничивается. При выделении заголовков с помощью символа («#») используется от одного до шести данных символов, которые устанавливаются в начале строки (перед заголовком). В данном случае количество символов соответствует уровню заголовка. Кроме того, заголовок возможно снабдить закрывающимися символами («#»), хотя это и не является обязательным. Количество закрывающихся символов не обязано соответствовать количеству начальных символов. Уровень заголовка определяется по количеству начальных символов.



Блочные элементы

- Вариант 1

Заголовок первого уровня

=====

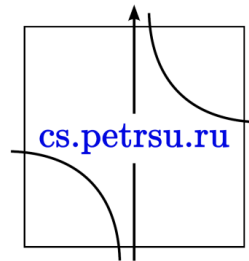
Заголовок второго уровня

- Вариант 2

Заголовок первого уровня

Заголовок третьего уровня

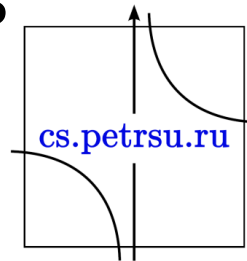
Заголовок шестого уровня



Блочные элементы

Цитаты

- Для обозначения цитат в языке Markdown используется знак «больше» («>»). Его можно вставлять как перед каждой строкой цитаты, так и только перед первой строкой параграфа. Также синтаксис Markdown позволяет создавать вложенные цитаты (цитаты внутри цитат). Для их разметки используются дополнительные уровни знаков цитирования («>»). Цитаты в Markdown могут содержать всевозможные элементы разметки.



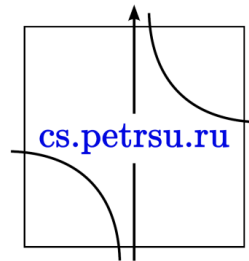
Примеры

1. Проводник
2. Полупроводник
3. Диэлектрик

- * Проводник
- * Полупроводник
- * Диэлектрик

- Проводник
- Полупроводник
- Диэлектрик

- + Проводник
- + Полупроводник
- + Диэлектрик



Примеры

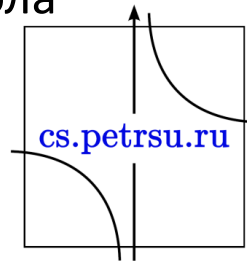
1. Элемент списка с цитатой:

- > Это цитата
- > внутри элемента списка.

2. Второй элемент списка

- При вставке цитат в элементы списка важно учитывать, что элементы списка должны находиться на одном уровне, а цитаты должны указываться с отступом. В случае, если правило с единым уровнем списка не соблюдается, следующий после цитаты элемент списка будет автоматически нумероваться цифрой «1.». Кроме того, при необходимости в список можно вставить исходный код. В этом случае его нужно писать с двойным отступом – 8 пробелов или 2 символа табуляции.

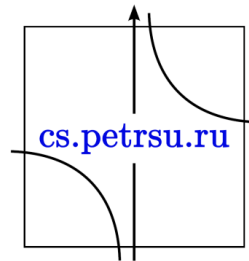
<исходный код >



Блочные элементы

Блоки кода

- Отформатированные блоки кода используются в случае необходимости процитировать исходный код программ или разметки. Для создания блока кода в языке Markdown необходимо каждую строку параграфа начинать с отступа, состоящего из четырех пробелов или одного символа табуляции. Блок кода продолжается до тех пор, пока не встретится строка без отступа (или конец текста). Внутри блока кода амперсанды («&») и угловые скобки («<» и «>») автоматически преобразуются в элементы HTML разметки. Кроме того, следует отметить, что внутри блоков кода обычный синтаксис Markdown не обрабатывается.



Блочные элементы

Горизонтальные линии (разделители)

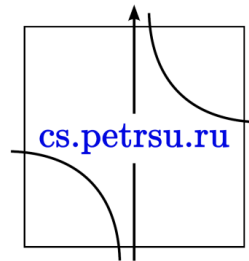
- Для того чтобы создать горизонтальную линию с использованием синтаксиса языка Markdown, необходимо поместить три (или более) дефиса или звездочки на отдельной строке текста. Между ними возможно располагать пробелы.

Первая часть текста, который необходимо разделить

Вторая часть текста, который необходимо разделить

Первая часть текста, который необходимо разделить

Вторая часть текста, который необходимо разделить



Строчные элементы

Ссылки

Markdown поддерживает два стиля оформления ссылок:

- Гиперссылка, с немедленным указанием адреса (внутритекстовая);
- Гиперссылка, подобная сноске.

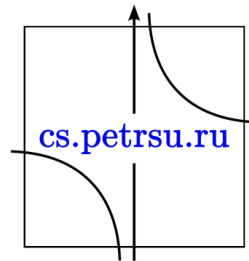
Подразумевается, что помимо URL-адреса существует еще текст ссылки. Он заключается в квадратные скобки. Для создания внутритекстовой гиперссылки необходимо использовать круглые скобки сразу после закрывающей квадратной. Внутри них необходимо поместить URL-адрес. В них же возможно расположить название, заключенное в кавычки, которое будет отображаться при наведении, но этот пункт не является обязательным.



Строчные элементы

Выделение текста

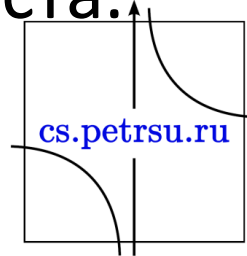
- Markdown воспринимает звёздочки «*» и символы подчёркивания «_» как признаки смыслового выделения текста:
- Текст, окружённый одиночными «*» или «_», будет заключен в HTML-тэг ``.
- Текст, окружённый двойными «*» или «_», будет заключен в HTML-тэг ``.



Строчные элементы

Кодовые фрагменты строк

- Чтобы отметить фрагмент строки, содержащий код, необходимо окружить его обратными апострофами «`». При использовании кодовых фрагментов строк текст будет отображаться в виде моноширинного шрифта. В отличие от блоков кода, кодовый фрагмент позволяет поместить код внутрь обычного абзаца текста.



Строчные элементы

Изображения

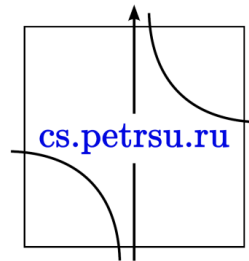
- С помощью непосредственного указания URL-адреса изображения.

```
![Альтернативный текст](/путь/к/изображению.jpg  
"Подсказка")
```

- С помощью метки-идентификатора.

```
![Альтернативный текст][id]  
[id]: путь/к/изображению "Необязательная подсказка«
```

- **Важно!** Markdown не позволяет задать размеры изображения (ширину, высоту).



Дополнительные элементы

- **Обратный слеш** - употребляться в Markdown перед специальными символами для того, чтобы они воспринимались в их буквальном (а не служебном) значении.

«\» - слеш;

«`» - обратный апостроф;

«*» - звездочка;

«_» - символ подчеркивания;

«{}» - фигурные скобки;

«[]» - квадратные скобки;

«()» - круглые скобки;

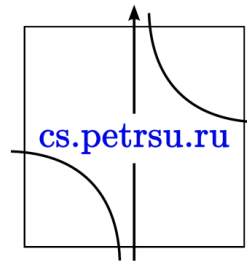
«#» - символ решетки;

«+» - плюс;

«-» - минус (дефис);

«.» - точка;

«!» - восклицательный знак.

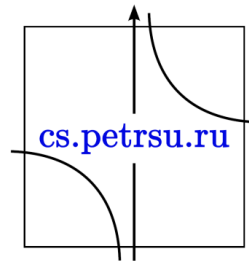


Автоматические ссылки

- Markdown поддерживает упрощённый порядок автоматического создания ссылок для URL-адресов и адресов электронной почты. Для этого достаточно поместить URL-адрес или почтовый адрес в угловые скобки, и Markdown сделает его гиперссылкой. В отличие от вышеописанных стилей, в данном случае сам же URL-адрес или почтовый адрес становится и текстом гиперссылки. Автоматические ссылки на адреса электронной почты работают аналогично.

`<http://example.com/>`

`<address@example.com>`

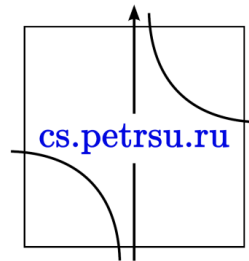


Обработка файла


- Pandoc <https://pandoc.org/>

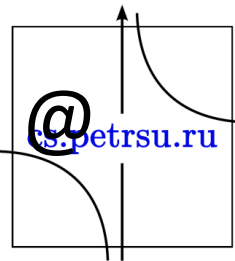
```
pandoc -s example30.docx -t markdown -o example35.md
```

```
pandoc example35.md -t odt --reference-doc=ref.odt -o example36.odt
```

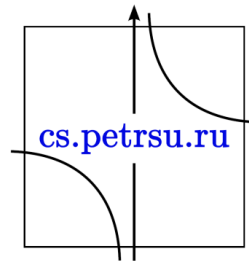


phpDocumentor

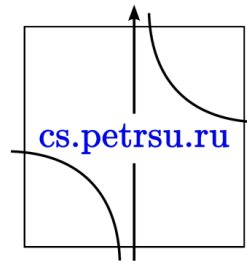
- Стандарт PHPDoc для документирования PHP-кода был реализован на основе уже существующего javaDoc для языка Java.
- Важной составляющей докблоков являются теги и аннотации, которые передают комментариям семантическую окраску.
- Тег или аннотация начинается с символа  www.petrso.ru



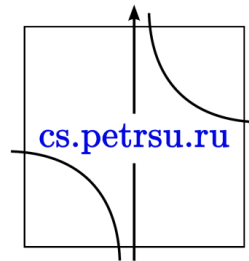
```
/**
 * Login via email and password
 *
 * @param Request $request Request
 *
 * @return Response
 *
 * @throws BadRequestHttpException
 * @throws UnauthorizedHttpException
 *
 * @Rest\Post("/login")
 */
public function loginAction(Request $request)
{
}
```



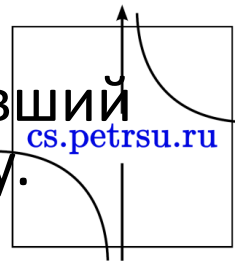
- Докблоки настолько прижились в коммюнити PHP-программистов, что на их основе готовится PSR-5 (PHP Standard Recommendation) PHPDoc Standard.
- Важно также, что один докблок может быть применён только к одному структурному элементу. Т.е. на каждую функцию — свой докблок, на переменную внутри функции — свой, для класса — свой.



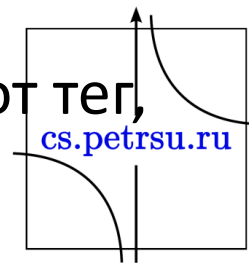
- В PHP с помощью докблоков можно документировать такие элементы:
 - функции;
 - константы;
 - классы;
 - интерфейсы;
 - трейты;
 - константы классов;
 - свойства;
 - методы.



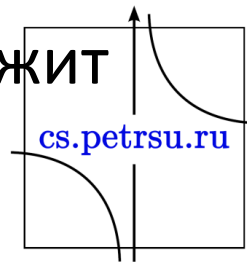
- **@api** (метод) — обозначает стабильные публичные методы, которые не будут менять свою семантику до следующего мажорного релиза.
- **@author** (в любом месте) — указывает имя и имейл автора, который написал следующий код.
- **@copyright** (в любом месте) — используется, чтоб поставить свой копирайт в коде.
- **@deprecated** (в любом месте) — полезный тег, символизирует, что данный элемент исчезнет в будущих версиях. Обычно рядом пишут, какой код следует использовать взамен. Также большинство IDE подсвечивают использование устаревших методов отдельным стилем. Когда нужно подчистить устаревший код для нового релиза, то легко искать по этому тегу.



- **@example** (в любом месте) — используется для размещения ссылки на файл или веб-страницу, где показан пример использования кода. На данный момент phpDocumentor заявляет о неполной поддержки возможностей этого тега.
- **@filesource** (файл) — этот тег можно размещать только на самом начале php-файла, так как тег применим только к файлу и включит весь код файла в сгенерированную документацию.
- **@global** (переменная) — на данный момент этот тег не поддерживается, возможно, будет реализован в следующих версиях, когда он будет переосмыслен.
- **@ignore** (в любом месте) — докблок, где указан этот тег, не будет обрабатываться во время генерации документации, даже если в нем есть другие теги.



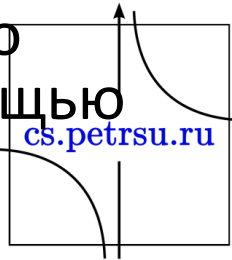
- **@internal** (в любом месте) — чаще всего используется вместе с тегом **@api**, чтоб показать, что код предназначен для внутренней логики этой части программы. Элемент, обозначенный этим тегом, не будет включен в документацию.
- **@license** (файл, класс) — что же он еще может делать, если не указывать тип лицензии для написанного кода.
- **@link** (в любом месте) — используется для вставки ссылок, но, как пишет документация, полностью функциональность тега пока не поддерживается.
- **@method** (класс) — применяется к классу и служит для описания магических методов, которые обрабатываются магической функцией `__call()`.



- **@package** (файл, класс) — разбиение кода на логические подгруппы. Когда вы помещаете классы в один namespace, вы тем самым показываете их функциональную схожесть. Если классы лежат в разных namespace'ах, но имеют одинаковый логический признак, их можно сгруппировать с помощью этого тега, например, если у вас классы работающие с корзиной заказа разбросаны по разным местам. Но лучше отказаться от такой практики, по код стайлу Symfony, например, этот тег не должен использоваться.
- **@param** (метод, функция) — предназначен для описания входящих параметров функции. Важно также отметить, что если вы уже взялись описывать входящие параметры для конкретной функции через докблоки, то нужно описывать все, а не только первый или второй.



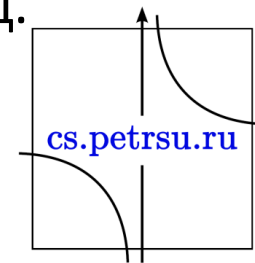
- **@property** (класс) — так же, как и **@method**, этот тег размещается в докблоке для класса, но описывает свойства, доступ к которым будет обрабатываться через магические методы `__get()` и `__set()`.
- **@property-read**, **@property-write** (класс) — аналогично предыдущему тегу, но обрабатывают только один магический метод, `__get()` или `__set()` соответственно.
- **@return** (метод, функция) — предназначен для описания значения, которое возвращает функция. Можно указать его тип, и например PhpStorm подхватит его и будет выдавать подсказки.
- **@see** (в любом месте) — с помощью этого тега можно вставлять ссылки на внешние ресурсы, как и с помощью **@link**, но также вставлять относительные ссылки на классы и методы.



- **@since** (в любом месте) — можно указать версию, в которой появился кусок кода.
- **@source** (в любом месте, кроме начала файла) — с помощью этого тега можно помещать в документацию участки исходного кода (задается строка начала и конца).
- **@throws** (метод, функция) — используется для указания исключений, которые могут быть вызваны в данной функции.
- **@todo** (в любом месте) — самый оптимистически тег, используется программистами, чтоб напомнить себе доделать что-то, когда-то в каком-то участке кода. IDE умеют распознавать этот тег и группируют все участки кода в отдельном окне, удобно для будущего поиска. Это общепринятый стандарт и используется очень часто.

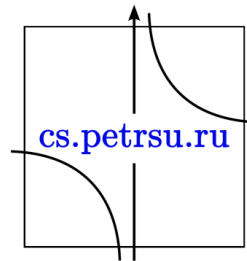


- **@uses** (в любом месте) — предназначен для отображения связи между разными участками кода. Он чем-то похож на **@see**, но разница в том, что **@see** создает однонаправленную ссылку, т.е. после перехода на новую страницу документации у вас не будет ссылки назад, а **@uses** в процессе его обработки ставит обратную ссылку, т.е. ссылку для обратной навигации.
- **@var** (переменная) — используется для указания типа и описания переменных, как тех, что встречаются внутри функций, так и свойств класса. Следует учесть разницу с тегом **@param**. Тег **@param** используется только в докблоках для функций и описывает входящие параметры, а **@var** используется для документирования обычных переменных.
- **@version** (в любом месте) — обозначает текущую версию программы, в которой появился данный класс, метод и т.д.



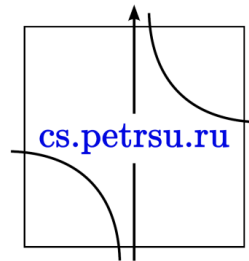
Правила

- Соблюдение правил документирования всеми участниками разработки.
- Использовать стиль кодирования включающий в себя докблоки.
- Если есть возможность IDE – включить инспектирование докблоков.
- Если есть возможность в IDE – включить автогенерацию докблоков.



Генерация документации

```
$ wget http://www.phpdoc.org/phpDocumentor.phar  
$ chmod +x phpDocumentor.phar  
$ sudo mv phpDocumentor.phar /usr/local/bin/phpdoc  
$ phpdoc --version  
$ phpdoc -d src/
```



Другие инструменты

- JavaDoc
- Epydoc 3.0 (2008)
- PyDoc
- Pyment
- Sphinx
- [Natural Docs](#) 2.2 (C# и др.)
- Плагины:
 - [AutoDocstring](#) – для VS Code.
 - [AutoDocstring](#) – для SublimeText.
 - [Python DocBlock Package](#) – для Atom.
 - [Autodoc](#) – для PyCharm.
- 11

