© Библиотека для взаимодействия с API проверки уникальности текстов сервиса text.ru

Все описываемые ниже классы пакета располагаются в пространстве имен `TextMedia\PlagiarismApi`, поэтому далее это не упоминается, и адресация классов ведётся относительно этого значения.

Установка

Для работы библиотеки требуется `php` версии 7.0 и выше и наличие расширения json (https://www.php.net/manual/ru/book.json.php) (включено по умолчанию, начиная с версии `PHP 5.2.0`); все нижеперечисленные методы установки одинаково работоспособны, т.к. пакет не имеет каких-либо других зависимостей.

Из `packagist`

```bash
cd my-project-folder
composer require text-media/plagiarism-api
```

Далее в своем скрипте подключаем автозагрузчик классов:

```php
require_once(__DIR__ . '/vendor/autoload.php');
```

Этот метод единственно гарантирует стабильность работы, т.к. в Packagist (https://packagist.org/packages/text-media/plagiarism-api) публикуются только стабильные версии, при использовании же нижеследующих способов можно получить нестабильную версию, находящуюся на стадии разработки/тестирования, либо же версия будет устаревшая, не включающая в себя весь функционал.

Обновление пакета до последней версии:

```bash
cd my-project-folder
composer update text-media/plagiarism-api
```

Установка `composer` - см. Download Composer (https://getcomposer.org/download/); например:

```bash
wget -O composer-setup.php https://getcomposer.org/installer
sudo php composer-setup.php --install-dir=/bin --filename=composer
rm -f composer-setup.php
```

Из репозитория

```bash
cd my-project-folder
git clone git@bitbucket.org:text-media/plagiarism-api.git
```

Добавить автозагрузчик классов в свой скрипт:

```php
require_once "{$my_project_folder}/plagiarism-api/autoload.php";
```

Обновление:

```bash
cd my-project-folder/plagiarism-api
git pull
```

Из архива

- на странице загрузок (https://bitbucket.org/text-media/plagiarism-api/downloads/?tab=tags) выбрать последюю версию архива репозитория;
- скачать и распаковать в корень проекта архив;
- переименовать полученную папку в `plagiarism-api`;
- в своем `php`-скрипте подключить автозагрузчик (см. предыдущий способ).

Обновление: повторить все перечисленные выше шаги (кроме последнего пункта).

Из `phar`

- Открыть файл bin/tmpa.phar (https://bitbucket.org/text-media/plagiarism-api/src/master/bin/tmpa.phar).
- В выпадающем списке фильтра выбрать раздел "Метки" - в нем последнюю версию репозитория.
- Нажать ссылку "Посмотреть исходный код" и сохранить файл.
- Добавить автозагрузчик классов в свой скрипт:

```php
require_once "phar://{$path_to_phar}/tmpa.phar";
```

Обновление:

```bash
TMPA=cli /usr/bin/env php /path/to/tmpa.phar self_update
```

Данный архив так же может использоваться для отправки `API` запросов из командной строки (см. Консоль (#23)).

Для работы потребуются два расширения `PHP`: phar (https://www.php.net/manual/ru/book.phar.php) и bz2 (https://www.php.net/manual/ru/book.bzip2.php).

На `php5`

Библиотека написана под `php` версии `7.0` и выше. Тем не менее, она способна работать и с версией `5.x`. Это возможно только, если устанавливать ее любым из перечисленных выше способов, кроме из `packagist`:

- из репозитория: дополнительно потребуется переключиться на нужную ветку: `git checkout php5`;
- из архива: нужно выбрать раздел "Ветки" и в нём - `php5`;
- из `phar`: по прямой ссылке (https://bitbucket.org/text-media/plagiarism-api/raw/php5/bin/tmpa.phar); **важно!**: нельзя будет использовать команду `self_update`, т.к. это приведет к обновлению до архива, предназначенного под версию `7.0` и выше.

Установка при помощи composer (https://getcomposer.org/download/) так же возможна, но для этого потребуется вручную отредактировать `composer.json` в корне проекта, чтобы он включал в себя следующие настройки:

```json
{
    "repositories": [
        {
            "type": "git",
            "url":  "git@bitbucket.org:text-media/plagiarism-api.git"
        }
    ],
    "require": {
        "text-media/plagiarism-api": "dev-php5"
    }
}
```

Применение

Проверка баланса пакетов

Внимание! В целях безопасности, чтобы ваш IP-адрес не заблокировался, не рекомендуется обращаться к данному методу чаще 2 раз в минуту.

```php
use Exception;
use TextMedia\PlagiarismApi\Client;
use TextMedia\PlagiarismApi\Exception\ClientException;
use TextMedia\PlagiarismApi\Exception\RequestException;
use TextMedia\PlagiarismApi\Exception\ResponseException;
use TextMedia\PlagiarismApi\Exception\ServerException;
use TextMedia\PlagiarismApi\Exception\TransportException;

try {
    $client   = new Client('my-api-key');
    $response = $client->getPackageBalance();
    echo "Баланс: {$response->getSize()}\n";
} catch (ServerException $ex) {
    echo "Ошибка API: [{$ex->getCode()}] {$ex->getMessage()}.\n";
    if ($ex->requiresToStop()) {
        echo "Нужно прекратить выполнять любые запросы к серверу.\n";
    } elseif ($ex->requiresToWait()) {
        echo "Нужно подождать какое-то время.\n";
    } elseif ($ex->requiresToSkip()) {
        echo "Нужно прекратить пытаться выполнять данный запрос.\n";
    } else {
        echo "Прочие ошибки сервера.\n";
    }
} catch (ClientException $ex) {
    echo "Неправильно настроен клиент API.\n";
} catch (TransportException $ex) {
    echo "Либо неправильно настроен транспорт до сервера API,";
    echo "либо возможно просто сервер временно недоступен";
    echo "(ошибка обмена данными) - попробуем позже еще раз.\n";
} catch (RequestException $ex) {
    echo "Неправильно сформирован запрос - требуется правка скрипта!\n";
} catch (ResponseException $ex) {
    echo "Что-то не то с ответом от API - требуется отладка!\n";
} catch (Exception $ex) {
    echo "Все прочие ошибки, не связанные непосредственно с пакетом.\n";
}
```

Данный шаблон можно использовать для всех видов запросов для правильной реакции на ответы/ошибки, поэтому в последующих примерах все перечисленные операции перехвата исключений и проверки ошибок опускаются - даются только примеры отправки запросов и анализа ответов на них в случае успеха.

Отправка текста на проверку

```php
use TextMedia\PlagiarismApi\Client;
use TextMedia\PlagiarismApi\Request;

$client = new Client('my-api-key');
$result = $client->sendTextToCheck([
    Request::OPT_TEXT           => file_get_contents('/path/to/file.txt'),
    Request::OPT_EXCEPT_DOMAINS => [
        'my.site.tld',
        'www.my.site.tld',
    ],
]);

echo "UID текста: {$result->getTextUid()}\n";
```

Получение результата проверки

```php
use TextMedia\PlagiarismApi\Client;
use TextMedia\PlagiarismApi\Request;

$client = new Client('my-api-key');
$result = $client->getPlagiarismResult([
    Request::OPT_TEXT_UID        => 'text-uid',
    Request::OPT_DETAILED_RESULT => true,
]);

echo "Уникальность текста: {$result->getUniqueness()}%\n";

if ($result->isSeoAnalyzeComplete()) {
    $seo = $result->getSeoAnalyzeDetail();
    echo "Процент воды: {$seo->getWaterPercent()}%\n";
    echo "Процент спама: {$seo->getSpamPercent()}%\n";
} else {
    echo "SEO-анализ текста еще не выполнен.\n";
}

if ($result->isSpellingComplete()) {
    $spell = $result->getSpellingDetail();
    echo "Обнаружено орфографических ошибок: " . count($spell) . "\n";
} else {
    echo "Проверка орфографии еще не завершена.\n";
}
```

Клиент

Класс: `Client`.

Метод	Описание
`__construct(string $userKey = null, Transport\TransportInterface $transport = null)`	Конструктор
`Response\ResponseInterface sendRequest(Request\RequestInterface $request)`	Выполнение произвольного запроса и получение ответа на него
`Response\CheckTextResponse sendTextToCheck(mixed $data)`	Получение результата на запрос "отправить текст на проверку"
`Response\PlagiarismResultResponse getPlagiarismResult(mixed $data)`	Получение результата на запрос "получить результат проверки текста"
`Response\PackageBalanceResponse getPackageBalance()`	Получение результата на запрос "получить информацию по пакетам"

Оба аргумента конструктора опциональны:

- `$userKey` может быть взят из переменной окружения `TMPA_USER_KEY` (см. putenv() (https://www.php.net/manual/ru/function.putenv.php));
- `$transport` определяется автоматически методом `Transport\AbstractTransport::getDefault()`.

Параметр `$data` методов `sendTextToCheck` и `getPlagiarismResult` могут быть либо объектом-запросом соответствующего класса, либо массивом для формирования такого объекта.

Транспорт

Интерфейс: `Transport\TransportInterface`.

Метод	Описание
`__construct(integer $timeout = 10, boolean $debugMode = false)`	Конструктор
`string process(string $entryPoint, array $request)`	Отправка запроса - возврат ответа
`self setTimeout(integer $timeout)`	Изменяет значение таймаута
`integer getTimeout()`	Возвращает значение таймаута
`static void check()`	Определяет доступен ли данный вид транспорта

Классы данного типа предназначены осуществлять обмен данными с `API`, т.е. собственно отправлять запрос, получать и возвращать ответ.

Пакет включает в себя три реализации транспорта:

- `Transport\StreamTransport`: запросы выполняются при помощи функций для работы с потоками (https://www.php.net/manual/ru/ref.stream.php); требует включенения директивы allow_url_fopen (https://www.php.net/manual/ru/filesystem.configuration.php);
- `Transport\CurlTransport`: требует наличия установленного расширения cURL (https://www.php.net/manual/ru/book.curl.php);
- `Transport\SocketTransport`: требует наличия установленного расширения sockets (https://www.php.net/manual/ru/book.sockets.php).

Метод `Transport\AbstractTransport::getDefault($timeout)` сам определит, какой из перечисленных выше транспортов доступен и вернет его (перебирает в том же порядке до тех пор, пока не найдет подходящий); если не найдет - будет выброшено исключение `Exception\TransportException`.

Запросы

Интерфейс: `Request\RequestInterface`.

Метод	Описание
`__construct(array $rawData = null)`	Конструктор
`array getRawData()`	Получение данных в "голом" виде, т.е. в том, в котором они отправляются на `API`

Формат описания запросов (полей массива, передаваемого в конструктор):

- имя поля (константа класса `Request`) в массиве, передаваемом в конструктор;
- тип данных;
- указание, является ли опция обязательной (+/−);
- описание собственно параметра, т.е. для чего служит.

Проверка баланса пакетов

Класс: `Request\PackageBalanceRequest`.

Никаких параметров не требует.

Отправка текста на проверку

Класс: `Request\CheckTextRequest`.

Поле	Тип/ Обязательность		Описание
`OPT_TEXT`	`string`	+	Проверяемый текст
`OPT_EXCEPT_DOMAINS`	`string|array`	−	Домены, которые вы хотите исключить из проверки
`OPT_EXCEPT_URLS`	`string|array`	−	Ссылки, которые вы хотите исключить из проверки
`OPT_VISIBILITY`	`boolean`	−	Включение доступности результатов проверки другим пользователям
`OPT_ARCHIVE`	`boolean`	−	Отключение сохранения результата проверки в архиве (ссылок вида `https://text.ru/antiplagiat/{text_uid}`)
`OPT_CALLBACK`	`string`	−	Callback (#22): URL (ссылка), на которую будет отправлен POST-запрос с результатами проверки

Получение результата проверки

Класс: `Request\PlagiarismResultRequest`.

Поле	Тип/Обязательность		Описание
`OPT_TEXT_UID`	`string`	+	Уникальный идентификатор текста
`OPT_DETAILED_RESULT`	`boolean`	−	Требование получить более детальную информацию о результатах проверки

Ответы

Интерфейс: `Response\ResponseInterface`.

Метод	Описание
`__construct(string|array $rawData = null)`	Конструктор
`array getRawData()`	Получение данных в "голом" виде, т.е. в том, в котором они получены от `API`

Проверка баланса пакетов

Класс: `Response\PackageBalanceResponse`.

Метод	Описание
`integer getSize()`	Суммарное число доступных для использования символов во всех имеющихся пакетах

Отправка текста на проверку

Класс: `Response\CheckTextResponse`.

Метод	Описание
`string getTextUid()`	Уникальный идентификатор текста

Получение результата проверки

Класс: `Response\PlagiarismResultResponse`.

Метод	Описание
`string|NULL getTextUid()`	UID текста (для запросов, пришедших в Callback (#22)).
`Response\Detail\PercentDetail getUniqueness()`	Уникальность текста в процентах с точностью до 2 знаков после запятой
`Response\Detail\PlagiarismDetail getPlagiarismDetail()`	Дополнительная информация о результатах проверки на уникальность
`Response\Detail\SpellingDetail[]|NULL getSpellingDetail()`	Дополнительная информация о результатах проверки на правописание
`Response\Detail\SeoAnalyzeDetail|NULL getSeoAnalyzeDetail()`	Дополнительная информация о результатах проверки на SEO-анализ
`boolean isSpellingComplete()`	Завершена ли проверка оффографии?
`boolean isSeoAnalyzeComplete()`	Завершен SEO-анализ?

Детальная информация по результатам проверки (последние два метода и некоторые данные во втором) возвращается только при указании опции `OPT_DETAILED_RESULT` в запросе. Если опция была указана, но методы все еще возвращают `NULL` - значит соответствующие проверки еще не завершились, нужно подождать и отправить запрос повторно.

Сложные структуры данных ответа оборачиваются в специальные классы из пространтва имен `Response\Detail`, которые являясь по факту частью ответа, содержат в себе только геттеры.

Общие методы (кроме `Response\Detail\PercentDetail`):

Метод	Описание
`__construct(string|array $rawData)`	Конструктор
`array getRawData()`	Получение данных в "голом" виде, т.е. в том, в котором они получены от `API`

Информация об уникальности

Класс: `Response\Detail\PlagiarismDetail`.

Метод	Описание
`\Datetime getDateTime()`	Дата окончания проверки текста на сервере.
`Response\Detail\PercentDetail getUniqueness()`	Уникальность текста в процентах с точностью до 2 знаков после запятой.
`integer[] getMixedWords()`	Номера слов (из очищенного текста), которые содержат одновременно символы из нескольких разных алфавитов.
`Response\Detail\MatchLinkDetail[] getMatchLinks()`	Найденные ссылки и процент совпадения с текстами по ним.
`string|string[]|NULL getClearText(boolean $cut = false)`	Очищенный от служебных символов и знаков препинания текст, состоящий из слов, разделенных через пробел.

Очищенный текст (последний метод) возвращается только при указании опции `OPT_DETAILED_RESULT` в запросе. Параметр `$cut` этого метода отвечает за то, будет ли возвращена просто строка или массив слов.

Процент совпадения со ссылкой

Класс: `Response\Detail\MatchLinkDetail`.

Метод	Описание
`mixed getUrl(integer $component = null)`	Собственно ссылка
`Response\Detail\PercentDetail getMatch()`	Процент совпадения текста по ссылке
`integer[]|NULL getWords()`	Номера совпавших слов (из очищенного текста; нумерация - от 0)

- Если параметр `$component` в методе `getUrl` будет пропущен - будет возвращен собственно `URL`; в противном случае метод отреагирует на него аналогично функции parse_url (http://php.net/manual/ru/function.parse-url.php), т.е. возвращена будет часть `URL` (или `FALSE`, если компонент указан неправильно или отсутствует).
- Номера совпавших слов (последний метод) возвращается только при указании опции `OPT_DETAILED_RESULT` в запросе.

Информация о правописании

Класс: `Response\Detail\SpellingDetail`.

Метод	Описание
`string getType()`	Тип ошибки ('Орфография', 'Пунктуация' и т.д.)
`string getDescription()`	Детальное описание ошибки
`string getText()`	Текст фрагмента, в котором обнаружилась ошибка
`string[] getReplacements()`	Массив с предлагаемыми вариантами замены (может быть пустым)
`integer getTextStart()`	Начальная позиция фрагмента, в котором найдена ошибка
`integer getTextEnd()`	Конечная позиция фрагмента, в котором найдена ошибка

Информация по SEO-анализу

Класс: `Response\Detail\SeoAnalyzeDetail`.

Метод	Описание
`integer getCountChars(TRUE)`	Количество символов с пробелами
`integer getCountChars(FALSE)`	Количество символов без пробелов
`integer getCountWords()`	Количество слов в тексте
`Response\Detail\PercentDetail getWaterPercent()`	Процент "воды" в тексте
`Response\Detail\PercentDetail getSpamPercent()`	Процент заспамленности
`Response\Detail\TextKeyDetail[] getListKeys(boolean $groupped)`	Список (простых и сгруппированных по составу слов) ключей в тексте

- Параметр `$groupped` метода `getListKeys` отвечает за то, какой список будет отдан - простых ключей или сгруппированных.
- Список простых ключей сортируется по частоте их встречаемости в тексте; сгруппированных - по числу значимых слов.

Информация о ключе в тексте.

Класс: `Response\Detail\TextKeyDetail`.

Метод	Описание
`integer getCount()`	Количество вхождений ключа в текст во всех формах
`string getTitle()`	Текст ключа (слово и фраза)
`Response\Detail\TextKeyDetail[]|NULL getSubKeys()`	Список простых подключей
`boolean isGroupped()`	Является ли данный ключ сгруппированным по составу слов

Ключи бываают простыми (состоят из одного слова) и сгруппированными по составу слов (представляют собой фразу). Список простых подключей имеется только в сгруппированном; в простом метод `getSubKeys` вернет `NULL`.

Процентные значения

Класс: `Response\Detail\PercentDetail`.

Метод	Описание
`__construct(mixed $value, string $type)`	Конструктор
`string getValue()`	Получение собственно значения процента
`string getType()`	Получение типа процента (уникальность, "вода", спам)
`string getClass()`	Класс значения процента: "bad", "good", "average"
`boolean isGood()`	Является ли данный процент хорошим значением для данного типа
`boolean isBad()`	Является ли данный процент плохим значением для данного типа
`boolean isAverage()`	Является ли данный процент средним значением для данного типа

В этот класс оборачиваются все значения процентов чего бы то ни было. Доступные типы и их границы "плохой"/"хороший" перечислены в константах класса; границы аналогичны тем, которые используются на офсайте для подстветки результатов проверки:

Константа	bad	good	Описание
`TYPE_UNIQUE`	<70	≥90	Уникальность текста
`TYPE_WATER`	>30	≤15	Процент "воды" в тексте
`TYPE_SPAM`	>60	≤30	Процент заспамленности
`TYPE_PLAGIAT`	≥90	<20	Процент совпадения текстов

Исключения

В случае возникновения любых ошибок выбрасываются исключения:

Класс	Исключения, выбрасываемые при…
`Exception\ClientException`	создании клиента (обычно - неправильный `API_KEY`)
`Exception\TransportException`	транспорте запросов к `API` (сервер недоступен и т.п.)
`Exception\RequestException`	формировании запросов к `API` (неправильные параметры или их тип/формат/значение)
`Exception\ResponseException`	анализе ответов от `API` (неправильные параметры или их тип/формат/значение)
`Exception\ServerException`	получении читаемого ответа от `API`, но содержащего сообщение об ошибке на его стороне

Метод `getDebugInfo()` возвращает информацию о том, при обработке чего произошла ошибка.

`Exception\ServerException` имеет дополнительно три метода, проверяющие тип **Ошибки**:

Метод	Необходимо…
`boolean requiresToWait()`	просто подождать какое-то время, прежде чем повторно выполнять вызвавший её запрос (сервер не доступен, текст еще не проверен и т.п.)
`boolean requiresToSkip()`	прекратить выполнять вызвавший её запрос (текст пустой или слишком короткий/длинный, ошибка проверки и т.п.)
`boolean requiresToStop()`	прекратить выполнять любые запросы к серверу (не хватает символов в пакетах, неправильный ключ API и т.п.)

Коды ошибок на стороне сервера `API` (`$serverException->getCode()`) и их значение см. на странице описания API (https://api.text.ru/doc/api/ru) или в документе **doc/api/ru.md** репозитория.

Если текст ещё не проверен, и выброшено исключение `Exception\ServerException` с кодом 81, то для получения дополнительной информации доступны следующие два метода:

Метод	Описание
`integer getQueueSize()`	Размер очереди: сколько ещё тексто осталось перед тем, как начнется прооверка данного.
`integer getCompletionPercent()`	Процент завершенности: на сколько процентов выполнена проверка данного текста.

Callback

`Callback\CallbackTemplate` и `Callback\CallbackTrait` являются шаблонами основным методом которых является `process()`, принимающий необязательный параметр `$rawData`, и формирующий на основе его внутреннее свойство класса `Response\PlagiarismResultResponse`; если `$rawData` не передано, то автоматически по умолчанию данные берутся из `$_POST`.

Данные шаблоны имеют абстрактные `protected`-методы, которые должны реализовать их потомки:

Метод	Описание
`onReceipt()`	Действия, которые необходимо выполнить с полученными данными `Response\PlagiarismResultResponse $this->plagiarism`
`onError()`	Что нужно делать, если при выполнении предыдущего метода, было выброшено исключение `\Exception $this->exception`

Оба метода должны вернуть ответ типа:

Класс	Описание
`Callback\CallbackPassed`	Ответ "запрос успешно обработан"
`Callback\CallbackFailed`	Ответ "запрос обработан с ошибкой - требуется прекратить попытки" (нет такого текста, например)
`Callback\CallbackRetry`	Ответ "запрос обработан с ошибкой - требуется повторить попытку через указанное число секунд"

Конструкторы первых двух классов аргументов не имеют; последнего - число секунд, сколько нужно подождать перед повтором.

Внимание! На данный момент сервером `API` реализовано правильное поведение только на ответ типа `Callback\CallbackPassed`; остальные - считаются ошибкой принимающей стороны, т.е. будут выполнены повторные запросы.

Пример реализации класса-`callback`-а:

```php
use TextMedia\PlagiarismApi\Callback;

class                extends         \
{
    /**
     * Анализируем $this->plagiarism.
     */
    protected function onReceipt(): Callback\CallbackResponse
    {
        // … собственно анализ и попытка сохранения

        if (/* все нормально - запрос проанализирован и результат сохранен */) {
            return new Callback\CallbackPassed;
        } elseif (/* что-то случилось - нет такого текста, например */) {
            return new Callback\CallbackFailed;
        } else /* временные проблемы - нужно повторить запрос позднее */ {
            return new Callback\CallbackRetry(
                /* сколько секунд подождать до повтора запроса */
            );
        }
    }

    /**
     * Анализируем $this->exception и $this->rawData.
     */
    protected function onError(): Callback\CallbackResponse
    {
        // Проверяем что-то в переданных данных, исключении,
        // из-за чего возникла ошибка; логируем, если надо и т.п.
        // В конце - ответ по типу как в onReceipt.
    }
}
```

Пример контроллера в CodeIgniter (https://codeigniter.com/) - это может быть файл в папке `application/controller/` такого вида:

```php
class                extends
{
    public function __construct()
    {
        parent::__construct();
        (new MyTmauCallback)->process($this->input->post());
    }
}
```

Вместо создания двух раздельных классов - отдельно Callback-а, отдельно контроллера - можно скомбинировать их при помощи трейта (http://php.net/manual/ru/language.oop5.traits.php) `Callback\CallbackTrait` . В Symfony (https://symfony.com/) это может быть файл из папки `src/Controller/` вида:

```php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\Routing\Annotation\Route;
use TextMedia\PlagiarismApi\Callback;

class                        extends
{
    use Callback\CallbackTrait;

    /**
     * @param Symfony\Component\HttpFoundation\Request $request
     * @Route("/tmau_callback", methods={"POST"})
     */
    public function index(Request $request)
    {
        $this->process($request->request->all());
    }

    /**
     * Анализируем $this->plagiarism.
     */
    protected function onReceipt(): Callback\CallbackResponse
    {
        // … пример см. выше
    }

    /**
     * Анализируем $this->exception и $this->rawData.
     */
    protected function onError(): Callback\CallbackResponse
    {
        // … пример см. выше
    }
}
```

Консоль

Для этого понадобится `phar`-архив (см. установку **Из phar**).

Пример использования:

```bash
TMPA=cli /usr/bin/env php /path/to/tmpa.phar plagiarism_result\
    --userkey=my-api-key\
    --uid=my-text-uid\
    --jsonvisible=detail\
    --retun=result_json
```

Результат успешно выполненного запроса выводится в /dev/stdout (https://ru.wikipedia.org/wiki/Стандартные_потоки#Стандартный_вывод); тексты ошибок выводятся в /dev/stderr (https://ru.wikipedia.org/wiki/Стандартные_потоки#Стандартный_вывод_ошибок) в виде `<имя класса исключения>[<код исключения>]: <описание исключения>.`

Список доступных команд и их опций выводится при обращении к скрипту без параметров.