

📖 стандартный / **standard** Public

🌟 Руководство по стилю JavaScript с линтером и автоматическим исправлением кода

🔗 [standardjs.com](#)

📄 MIT license

⭐ 28k звезды 🍴 2.4k вилки

☆ Звезда

👁️ просмотр ▾

<> Код

🕒 Проблемы 87

🔗 Запросы


📄 на извлечение 12

🎬 Действия

🛡️ Безопасность

🔗 мастер ▾

⋮

 fawazahmed0 ...

❌ 6 февраля ⌚

Просмотр кода

☰ README.md



Стандартный стиль JavaScript

chat 10 online

🔄 Test External failing

🔄 Test Internal passing

🔄 Old test passing

npm v17.0.0

downloads 10M/month

code style standard

Спонсируется

 **Socket**

 **Wormhole**

Английский • Español (Latinoamérica) • Français • Bahasa Indonesia • Italiano
(итальянский) • Яп. 日 (японский) • 한국어 (корейский) • Português (Brasil) • 简体中文
(упрощенный китайский) • 繁體中文 (тайваньский мандарин)

Руководство по стилю JavaScript, линтер и средство форматирования

Этот модуль экономит ваше (и других!) Время тремя способами:

- **Нет конфигурации.**Самый простой способ обеспечить качество кода в вашем проекте. Никаких решений для принятия. Нет `.eslintrc` файлов для управления. Это просто работает.
- **Автоматическое форматирование кода.**Просто запустите `standard --fix` и попрощайтесь с беспорядочным или непоследовательным кодом.
- **Выявляйте проблемы со стилем и ошибки программиста на ранней стадии.**Экономьте драгоценное время на проверку кода, устраняя взаимные переписки между рецензентом и автором.

Попробуйте, запустив `npm run standard --fix` его прямо сейчас!

Содержание

- Быстрый старт
 - Установить
 - Использование
 - Что вы могли бы сделать, если вы умны
- Вопросы и ответы
 - Почему я должен использовать стандартный стиль JavaScript?
 - Кто использует стандартный стиль JavaScript?
 - Существуют ли плагины для текстового редактора?
 - Есть ли значок readme?
 - Я не согласен с правилом X, вы можете его изменить?
 - Но это не настоящий веб-стандарт!
 - Есть ли автоматический форматировщик?
 - Как мне игнорировать файлы?
 - Как мне отключить правило?
 - Я использую библиотеку, которая загрязняет глобальное пространство имен. Как мне предотвратить ошибки "переменная не определена"?
 - Как мне использовать экспериментальные функции JavaScript (ES Next)?

- Могу ли я использовать вариант языка JavaScript, например Flow или TypeScript?
- Как насчет Mocha, Jest, Jasmine, QUnit и т. Д.?
- Как насчет веб-работников и сервисных работников?
- В чем разница между предупреждениями и ошибками?
- Могу ли я проверить код внутри файлов Markdown или HTML?
- Есть ли Git pre-commit -хук?
- Как мне сделать вывод красочным и красивым?
- Есть ли Node.js API?
- Как мне внести свой вклад в StandardJS?

Установить

Самый простой способ использовать стандартный стиль JavaScript - установить его глобально как программу командной строки узла. Выполните следующую команду в терминале:

```
стандарт установки $ npm - глобальный
```

Или вы можете установить `standard` локально для использования в одном проекте:

```
$ npm install standard --save-dev
```

Примечание: Для выполнения предыдущих команд, [Node.js](#) и [npm](#) должен быть установлен.

Использование

После установки вы `standard` сможете использовать `standard` программу. Самым простым вариантом использования будет проверка стиля всех файлов JavaScript в текущем рабочем каталоге:

```
$ standard
Ошибка: используйте стандартный стиль JavaScript
lib /torrent.js:950:11: Ожидалось '===', а вместо этого увидел '=='.
```

Если вы установили `standard` локально, запустите `npx` вместо:

```
$ стандарт npx
```

При желании вы можете передать каталог (или каталоги), используя шаблон глобуса. Обязательно указывайте пути, содержащие шаблоны глобусов, чтобы они расширялись с помощью вместо вашей оболочки `standard` :

```
$ standard "src/util/**/*.js" "тест / **/*.js"
```

Примечание: по умолчанию `standard` будут искать все файлы, соответствующие шаблонам: `**/*.js` , `**/*.jsx` .

Что вы могли бы сделать, если вы умны

1. Добавьте его в `package.json`

```
{
  "name": "my-cool-package",
  "devDependencies": {
    "стандартный": "*"
  },
  "скрипты": {
    "тест": "стандартный && узел my-tests.js "
  }
}
```

2. Стил проверяется автоматически при запуске `npm test`

```
$ npm-тест
Ошибка: используйте стандартный стиль JavaScript
lib /torrent.js:950:11: Ожидалось '===', а вместо этого увидел '=='.
```

3. Никогда больше не давайте отзыв о стиле на запрос на извлечение!

Почему я должен использовать стандартный стиль JavaScript?

Прелесть стандартного стиля JavaScript в том, что он прост. Никто не хочет поддерживать несколько файлов конфигурации стиля в сто строк для каждого модуля / проекта, над которым они работают. Хватит этого безумия!

Этот модуль экономит ваше (и других!) Время тремя способами:








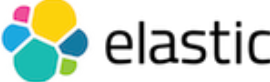






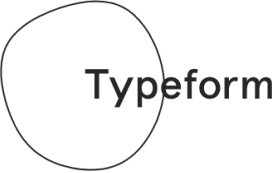









- **Нет конфигурации.** Самый простой способ обеспечить согласованный стиль в вашем проекте. Просто добавьте его.








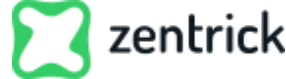










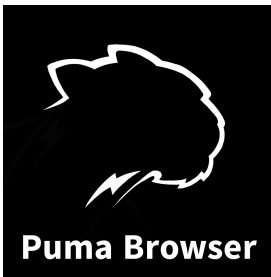











- **Автоматическое форматирование кода.** Просто запустите `standard --fix` и попрощайтесь с беспорядочным или непоследовательным кодом.
- **Выявляйте проблемы со стилем и ошибки программиста на ранней стадии.** Экономьте драгоценное время на проверку кода, устраняя взаимные переписки между рецензентом и автором.

Принятие `standard` стиля означает, что важность ясности кода и соглашений сообщества выше, чем личный стиль. Это может не иметь смысла для 100% проектов и культур разработки, однако открытый исходный код может быть враждебным местом для новичков. Настройка четких, автоматических ожиданий участников делает проект более здоровым.

Для получения дополнительной информации см. Доклад на конференции "Напишите идеальный код с помощью Standard и ESLint". В этом докладе вы узнаете о линтинге, когда использовать `standard` versus `eslint` и как `prettier` сравнивать с `standard`.

Кто использует стандартный стиль JavaScript?

Помимо компаний, многие члены сообщества используют standard пакеты npm, которых **слишком много**, чтобы перечислять здесь.

`standard` также является лучшим линтером на выставке [Clean Code Linter](#) на GitHub.

Существуют ли плагины для текстового редактора?

Сначала установите `standard`. Затем установите соответствующий плагин для вашего редактора:

Возвышенный текст

Используя [управление пакетами](#), установите [SublimeLinter](#) и [SublimeLinter-contrib-standard](#).

Для автоматического форматирования при сохранении установите [StandardFormat](#).

Atom

Установите [linter-js-standard](#).

В качестве альтернативы вы можете установить [linter-js-standard-engine](#). Вместо того, чтобы связывать его версию `standard`, автоматически будет использоваться версия, установленная в вашем текущем проекте. Он также будет работать "из коробки" с другими линтерами, основанными на [стандартном движке](#).

Для автоматического форматирования установите [standard-formatter](#). Для фрагментов установите [standardjs-snippets](#).

Visual Studio Code

Установите [vscode-standard](#). (Включает поддержку автоматического форматирования.)

Для фрагментов JS установите: [vscode-standardjs-snippets](#). Для фрагментов React установите [vscode-react-standard](#).

Vim

Установите [ale](#). И добавьте эти строки в свой `.vimrc` файл.

```
= g:ale_linters пусть {  
  \ 'javascript': ['standard'],  
  \  
  пусть g:ale_fixers = {'javascript': ['standard']}
```

Это устанавливает стандарт в качестве единственного линтера и исправителя для файлов javascript и, таким образом, предотвращает конфликты с eslint. Для линтинга и автоматического исправления при сохранении добавьте эти строки в .vimrc :

```
пусть g: ale_lint_on_save = 1
пусть g: ale_fix_on_save = 1
```

Альтернативные плагины, которые следует рассмотреть, включают [neomake](#) и [syntastic](#), оба из которых имеют встроенную поддержку standard (хотя может потребоваться настройка).

Emacs

Установите [Flycheck](#) и ознакомьтесь с [руководством](#), чтобы узнать, как включить его в своих проектах.

Скобки

Найдите в реестре расширений "[Стандартный стиль кода](#)" и нажмите "Установить".

WebStorm (PhpStorm, IntelliJ, RubyMine, JetBrains и др.)

WebStorm [недавно объявил о встроенной поддержке](#) standard непосредственно в IDE.

Если вы все еще предпочитаете настраивать standard вручную, [следуйте этому руководству](#). Это относится ко всем продуктам JetBrains, включая PhpStorm, IntelliJ, RubyMine и др.

Есть ли значок readme?

Да! Если вы используете standard в своем проекте, вы можете включить один из этих значков в свой readme, чтобы люди знали, что ваш код использует стандартный стиль.



[![Стиль JavaScript Guide](https://cdn.rawgit.com/standard/standard/master/badge

[! [Руководство по стилю JavaScript] (https://img.shields.io/badge/code_style-standard)

Я не согласен с правилом X, вы можете его изменить?

Нет. Весь смысл `standard` в том, чтобы сэкономить ваше время, избегая **путаницы** в стиле кода. В Интернете много споров о табуляции, пробелах и т. Д., Которые никогда не будут решены. Эти дебаты просто отвлекают от выполнения работы. В конце дня вам нужно "просто выбрать что-то", и в этом вся философия `standard` - это куча разумных мнений "просто выбери что-нибудь". Надеюсь, пользователи видят в этом ценность, а не отстаивают свое собственное мнение.

Есть пара похожих пакетов для тех, кто не хочет полностью принимать `standard` :

- **полустандартный** - стандартный, с точкой с запятой
- **standardx** - стандартный, с пользовательскими настройками

Если вы действительно хотите настроить сотни правил ESLint по отдельности, вы всегда можете использовать `eslint` **eslint-config-standard**, чтобы наложить свои изменения поверх. **standard-eject** может помочь вам перейти `standard` с `eslint` `eslint-config-standard`

Совет профессионала: просто используйте `standard` и двигайтесь дальше. Есть реальные реальные проблемы, на решение которых вы могли бы потратить свое время! : P

Но это не настоящий веб-стандарт!

Конечно, это не так! Стил, изложенный здесь, не связан ни с какими официальными группами веб-стандартов, поэтому это репозиторий называется `standard/standard`, а не `ECMA/standard`.

Слово "стандартный" имеет больше значений, чем просто "веб-стандарт" :-)
Например:

- Этот модуль помогает поддерживать наш код на высоком *уровне качества*.
- Этот модуль гарантирует, что новые участники следуют некоторым базовым *стандартам стиля*.

Есть ли автоматический форматировщик?

Да! Вы можете использовать `standard --fix` для автоматического устранения большинства проблем.

`standard --fix` встроен в `standard` для максимального удобства. Большинство проблем можно исправить, но некоторые ошибки (например, забывание обрабатывать ошибки) необходимо исправлять вручную.

Чтобы сэкономить ваше время, `standard` выводит сообщение "Run standard --fix to automatically fix some problems" при обнаружении проблем, которые могут быть устранены автоматически.

Как мне игнорировать файлы?

Определенные пути (`node_modules/`, `coverage/`, `vendor/`, `*.min.js`, и файлы / папки, начинающиеся с `.like .git/`) автоматически игнорируются.

Пути в корневом `.gitignore` файле проекта также автоматически игнорируются.

Иногда вам нужно игнорировать дополнительные папки или определенные уменьшенные файлы. Для этого добавьте `standard.ignore` свойство

в `package.json`:

```
"стандарт": {
  "игнорировать": [
    "**/out/",
    "/lib/select2/",
    "/lib/ckeditor/",
    "tmp.js "
  ]
}
```

Как мне отключить правило?

В редких случаях вам нужно будет нарушить правило и скрыть ошибку, сгенерированную `standard`.

Стандартный стиль JavaScript использует [ESLint](#) под капотом, и вы можете скрыть ошибки, как обычно, если бы вы использовали ESLint напрямую.

Отключите **все правила** для определенной строки:

```
file = 'Я знаю, что я делаю' // eslint-disable-line
```

Или отключить **только** "no-use-before-define" правило:

```
file = 'Я знаю, что я делаю' // eslint-отключить-строку, не используемую до оп
```

Или отключите "no-use-before-define" правило для **нескольких строк**:

```
/* eslint-отключить неиспользование перед определением */  
console.log('код-нарушитель идет сюда ...')  
console.log('код-нарушитель идет сюда ...')  
console.log('код-нарушитель идет сюда ...')  
/* eslint-включить неиспользование перед-определить */
```

Я использую библиотеку, которая загрязняет глобальное пространство имен. Как мне предотвратить ошибки "переменная не определена"?

Некоторые пакеты (например `mocha`) помещают свои функции (например `describe` , `it`) в глобальный объект (плохая форма!). Поскольку эти функции не определены или `require` не определены нигде в вашем коде, `standard` они будут предупреждать, что вы используете переменную, которая не определена (обычно это правило действительно полезно для обнаружения опечаток!). Но мы хотим отключить его для этих глобальных переменных.

Чтобы `standard` люди (а также люди, читающие ваш код) знали, что определенные переменные являются глобальными в вашем коде, добавьте это в начало вашего файла:

```
/* глобальные myVar1, myVar2 */
```

Если у вас сотни файлов, может быть желательно избегать добавления комментариев к каждому файлу. В этом случае запустите:

```
$ standard --глобальный myVar1 --глобальный myVar2
```

Или добавьте это в `package.json` :

```
{  
  "стандартный": {
```

```
"глобальные переменные": [ "myVar1", "myVar2"]
}
```

Примечание: `global` и `globals` эквивалентны.

Как мне использовать экспериментальные функции JavaScript (ES Next)?

`standard` поддерживает новейшие функции ECMAScript, ES8 (ES2017), включая предложения языковых функций, которые находятся на "стадии 4" процесса предложения.

Для поддержки экспериментальных языковых функций `standard` поддерживается указание пользовательского синтаксического анализатора JavaScript. Перед использованием пользовательского синтаксического анализатора подумайте, стоит ли того дополнительная сложность.

Чтобы использовать пользовательский анализатор, сначала установите его из npm:

```
npm install @babel/eslint-parser --save-dev
```

Затем запустите:

```
$ standard --parser @babel/eslint-parser
```

Или добавьте это в `package.json` :

```
{
  "стандартный": {
    "синтаксический анализатор": "@babel/ eslint-синтаксический анализатор"
  }
}
```

Могу ли я использовать вариант языка JavaScript, например Flow или TypeScript?

`standard` поддерживает новейшие функции ECMAScript. Однако Flow и TypeScript добавляют новый синтаксис в язык, поэтому они не поддерживаются "из коробки".

Для TypeScript `ts-standard` поддерживается и поддерживается официальный вариант, который обеспечивает очень похожий интерфейс `standard`.

Для других вариантов языка JavaScript `standard` поддерживается указание пользовательского синтаксического анализатора JavaScript, а также плагина ESLint для обработки измененного синтаксиса. Прежде чем использовать вариант языка JavaScript, подумайте, стоит ли того дополнительная сложность.

TypeScript

`ts-standard` является официально поддерживаемым вариантом для TypeScript. `ts-standard` поддерживает все те же правила и опции, `standard` что и и включает дополнительные правила, специфичные для TypeScript. `ts-standard` будет ли даже обычный `npm install ts-standard --save-dev`

```
npm install ts-standard --save-dev
```

Затем запустите (где `tsconfig.json` находится в рабочем каталоге):

```
$ ts-стандартный
```

Или добавьте это в `package.json`:

```
{
  "ts-standard": {
    "проект": "./tsconfig.json"
  }
}
```

Примечание: Чтобы включить в линтинг дополнительные файлы, такие как тестовые файлы, создайте `tsconfig.eslint.json` файл для использования вместо него.

Если вы действительно хотите настроить сотни правил ESLint по отдельности, вы всегда можете использовать `eslint` напрямую с `eslint-config-standard-with-typescript`, чтобы наложить свои изменения поверх.

Поток

Чтобы использовать Flow, вам нужно работать `standard` `@babel/eslint-parser` как синтаксический анализатор и `eslint-plugin-flowtype` как плагин.

```
npm install @babel/eslint-parser eslint-plugin-flowtype --save-dev
```

Затем запустите:

```
$ standard --parser @babel/eslint-parser --тип потока плагина
```

Или добавьте это в `package.json` :

```
{
  "стандартный": {
    "синтаксический анализатор": "@babel / eslint-parser",
    "плагины": [ "flowtype" ]
  }
}
```

Примечание: `plugin` и `plugins` эквивалентны.

Как насчет Mocha, Jest, Jasmine, QUnit и т. Д.?

Для поддержки mocha в тестовых файлах добавьте это в начало тестовых файлов:

```
/* eslint-env mocha */
```

Или запустите:

```
$ standard --env mocha
```

Где `mocha` может быть один из `jest`, `jasmine`, `qunit`, `phantomjs`, и так далее.

Чтобы увидеть полный список, ознакомьтесь с документацией ESLint, [определяющей среды](#). Список глобалов, доступных для этих сред, см. в модуле `npm globals`.

Примечание: `env` и `envs` эквивалентны.

Как насчет веб-работников и сервисных работников?

Добавьте это в начало файлов `web worker`:

```
/* eslint-env worker */
```

Это позволяет `standard` (а также людям, читающим код) знать, что `self` это является глобальным в коде `web worker`.

Для сервисных работников добавьте это вместо:

```
/ * eslint-env serviceworker */
```

В чем разница между предупреждениями и ошибками?

`standard` рассматривает все нарушения правил как ошибки, что означает, что `standard` будет выполнен выход с ненулевым (ошибочным) кодом выхода.

Тем не менее, мы можем иногда выпускать новую основную версию `standard`, в которой изменяется правило, влияющее на большинство `standard` пользователей (например, переход из `var` `let` / `const`). Мы делаем это только тогда, когда считаем, что преимущество стоит затрат, и только тогда, когда правило можно [исправить автоматически](#).

В таких ситуациях у нас есть "переходный период", когда изменение правила является всего лишь "предупреждением". Предупреждения не приводят `standard` к возврату ненулевого кода выхода (ошибки). Однако предупреждающее сообщение все равно будет выводиться на консоль. В течение переходного периода `using standard --fix` обновит ваш код, чтобы он был готов к следующей основной версии.

Медленный и тщательный подход - это то, к чему мы стремимся `standard`. Мы, как правило, крайне консервативны в обеспечении использования новых языковых функций. Мы хотим, чтобы использование `standard` было легким и увлекательным, поэтому мы тщательно относимся к внесению изменений, которые могут помешать вам. Как всегда, вы можете [отключить правило](#) в любое время, если это необходимо.

Могу ли я проверить код внутри файлов Markdown или HTML?

Чтобы проверить код внутри файлов Markdown, используйте `standard-markdown`.

Кроме того, существуют плагины ESLint, которые могут проверять код внутри Markdown, HTML и многих других типов языковых файлов:

Чтобы проверить код внутри файлов Markdown, используйте плагин ESLint:

```
$ npm установить eslint-плагин-markdown
```

Затем, чтобы проверить JS, который отображается внутри блоков кода, выполните:

```
$ standard --плагин markdown '**/*.md'
```

Для проверки кода внутри HTML-файлов используйте плагин ESLint:

```
$ npm установить eslint-plugin-html
```

Затем, чтобы проверить JS, который отображается внутри `<script>` тегов, выполните:

```
$ standard --плагин html '**/*.html'
```

Есть ли Git pre-commit -хук?

Да! Хуки отлично подходят для обеспечения того, чтобы нестайлинговый код никогда даже не попадал в ваш репозиторий. Никогда больше не давайте отзыв о стиле на запрос на извлечение!

У вас даже есть выбор...

Установите свой собственный хук

()

все файлы JavaScript, подготовленные для фиксации, соответствуют стандартному стилю версия "xargs -r". Флаг -r - это расширение GNU, которое предотвращает запуск xargs, если нет входных файлов.

```
find -r -d $ ' \ n' path; затем  
"$path" | cat - | xargs "$@"
```

```
ко для имен --кэшированный --относительный | grep '\.jsx\?$' | sed 's/^[[:alnum:]]  
е 0 ]]; затем  
ены ошибки стандартного стиля echo' JavaScript. Прерывание фиксации.'
```


Используйте pre-commit крюк

Библиотека [предварительной фиксации](#) позволяет объявлять перехваты в `.pre-commit-config.yaml` файле конфигурации в репозитории и, следовательно, ее легче поддерживать в команде.

Пользователи pre-commit могут просто добавить `standard` в свой `.pre-commit-config.yaml` файл, который автоматически исправит `.js`, `.jsx`, `.mjs` и `.cjs` файлы:

- репозиторий: `https://github.com/standard/standard`
 - rev: мастер
- крючки:
- идентификатор: стандартный

В качестве альтернативы, для более сложных конфигураций стиля, используйте `standard` в [eslint hook](#):

- репозиторий: `https://github.com/pre-commit/mirrors-eslint`
 - rev: мастер
- крючки:
- идентификатор: файлы `eslint`
 - : `\.[jt]sx?$` # `*.js`, `*.jsx`, `*.ts` и `*.tsx`
 - типы: `[file]`
 - additional_dependencies:
- `eslint@последняя версия`
- `eslint-config-standard@latest`
 - # и любые другие плагины ...

Как мне сделать вывод красочным и красивым?

Встроенный вывод прост и понятен, но если вам нравятся блестящие вещи, установите [snazzy](#):

```
$ npm установить шикарно
```

И запустите:

```
$ стандартный | шикарный
```

Существуют также [standard-tap](#), [standard-json](#), [standard-reporter](#) и [standard-summary](#).

Есть ли Node.js API?

Да!

async standard.lintText(text, [opts])

Удалите предоставленный исходный код. text opts Может быть предоставлен объект:

```
ИМЯ
  файла // уникальное для lintText
  {: ", // путь к файлу, содержащему текст, который нужно линтовать

  // общее для lintText и lintFiles
  cwd: ", // текущий рабочий каталог (по умолчанию: process.cwd())
  исправить: false, // автоматически исправлять проблемы
  расширения: [], // расширения файлов для lint (имеет нормальные значения)
  глобальные переменные: [], // пользовательские глобальные переменные
  плагинов: [], // пользовательские среды плагинов eslint
: [], // пользовательский анализатор среды eslint
: ", // пользовательский анализатор js (например, babel-eslint)
  usePackageJson: true, // использовать параметры из ближайшего package.json?
  useGitIgnore: true // использовать шаблоны игнорирования файлов из .gitignore
}
```

Все опции являются необязательными, хотя некоторые плагины ESLint требуют этой filename опции.

Дополнительные параметры могут быть загружены из a package.json , если он найден для текущего рабочего каталога. Смотрите ниже для получения дополнительной информации.

Возвращает Promise разрешение results или отклоняется с Error помощью .

results Объект будет содержать следующие свойства:

```
результаты { = results
const: [
  {
    filePath: ",
    сообщения: [
      { ruleId: ", message: ", строка: 0, столбец: 0 }
    ],
    количество ошибок: 0,
    warningCount: 0,
    вывод: " // исправлен исходный код (присутствует только с опцией {fix: true})
  }
]
```

```

],
  количество ошибок: 0,
  количество предупреждений: 0
}

```

async standard.lintFiles(files, [opts])

Удалите ворсинки из предоставленных files шариков. opts Может быть предоставлен объект:

```

игнорировать
  // уникальные для lintFiles
  {: [], // файловые глобусы для игнорирования (имеет нормальные значения)

  // общие для lintText и lintFiles
  cwd: "", // текущий рабочий каталог (по умолчанию: process.cwd())
  исправить: false, // автоматически исправлять проблемы
  расширения: [], // расширения файлов для lint (имеет нормальные значения)
  глобальные переменные: [], // пользовательские глобальные переменные
  плагинов: [], // пользовательские среды плагинов eslint
  : [], // пользовательский анализатор среды eslint
  : "", // пользовательский анализатор js (например, babel-eslint)
  usePackageJson: верно, // использовать параметры из ближайшего package.json
  useGitIgnore: верно // использовать шаблоны игнорирования файлов из .gitignore
}

```

Дополнительные параметры могут быть загружены из a package.json , если он найден для текущего рабочего каталога. Смотрите ниже для получения дополнительной информации.

Оба ignore files шаблона и разрешаются относительно текущего рабочего каталога.

Возвращает Promise разрешение results или отклоняется с Error помощью (то же, что и выше).

Как мне внести свой вклад в StandardJS?

Вклад приветствуется! Ознакомьтесь с проблемами или ссылками на источники и создайте свои собственные, если вам нужно что-то, чего вы там не видите.

Хотите пообщаться? Присоединяйтесь к участникам IRC в #standard канале на freenode.

Вот несколько важных пакетов в standard экосистеме:

- **стандарт** - это репозиторий
 - **стандартный движок** - движок cli для произвольных правил eslint
 - **eslint-config-standard** - правила eslint для стандартных
 - **eslint-config-standard-jsx** - правила eslint для стандарта (JSX)
 - **eslint** - линтер, который поддерживает стандарт
- **шикарно** - симпатичный вывод терминала для стандартного
- **стандарт-www** - код для <https://standardjs.com>
- **полустандартный** - стандартный, с точкой с запятой (если необходимо)
- **standardx** - стандартный, с пользовательскими настройками

Также есть множество **плагинов для редактора**, список **используемых пакетов npm standard** и потрясающий список **пакетов в standard экосистеме**.


Политики и процедуры безопасности

standard Команда и сообщество серьезно относятся ко всем ошибкам безопасности standard . Пожалуйста, ознакомьтесь с нашими **политиками и процедурами безопасности**, чтобы узнать, как сообщать о проблемах.

Лицензия

MIT. Авторское право (с) Феросс Абухадидже.

Выпуски 4

 последняя версия **17.0.0**
20 апреля 2022 года 20 апреля 2022

+ 3 выпуска

Спонсируйте этот проект



feross Феросс Абу Хадидже



стандарт Стандартный JS



tidelift.com/funding/github/npm/standard

Узнайте больше о спонсорах GitHub

Пакетов

Пакеты не опубликованы

Используется 195k



Авторы 175



+ 164 участника

Языки

