

СИСТЕМЫ УПРАВЛЕНИЯ ВЕРСИЯМИ*, GITHUB, GIT*, БЛОГ КОМПАНИИ EASTBANC TECHNOLOGIES

Невидимые друзья вашего github-репозитория

fshchudlo 13 марта 2017 в 12:11 👁 13,1k



Github это незаменимый инструмент, прочно вошедший в жизнь практически каждого разработчика.

Хотя многие из нас используют его постоянно, не все знают, что существует большое количество сторонних (и бесплатных) сервисов и инструментов, которые тесно интегрированы с github и расширяют его функциональность.

В данной статье мы уделим внимание, в основном, инструментам,

работающим в инфраструктуре npm. Полный список сервисов, интегрирующихся с github, можно посмотреть на странице [github integrations directory](#).

Сегодня в выпуске:

- Настраиваем continuous integration с travis ci
- Настраиваем отчеты о test coverage с coveralls
- Мониторим статус зависимостей с david
- Настраиваем автоматическое обновление зависимостей с greenkeeper
- Улучшаем сообщения коммитов с commitizen
- Генерируем changelog и release notes с conventional-changelog
- Управляем задачами с zube

Большинство рассмотренных сервисов предоставляет информационные значки (badges), которые можно добавить на страницу проекта. Чтобы не показывать одну и ту же картинку при каждом упоминании badge, покажем ее один раз в начале статьи:

e2e4

build **passing** coverage **100%** commitizen **friendly** dependencies **none** devDependencies **up to date** npm package **2.0.0-rc.4**

Set of base classes and utilities to build unobtrusive list models. This is abstract codebase which can be used to implement bridges to end-user frameworks (such as [Angular bridge](#)).

Documentation

Documentation is available [here](#)

How to build the project

To build the project, follow these steps:

1. Ensure that [NodeJS](#) is installed. This provides the platform on which the build tooling runs.

Также нам понадобится написать несколько скриптов. Для этого мы будем использовать npm script-ы, избегая таких инструментов как grunt или gulp, так что знание специфических сборщиков вам не понадобится.

Небольшое лирическое отступление про task runners

Живые примеры использования описанных сервисов можно посмотреть в наших репозиториях библиотеки [e2e4](#) (простые варианты) или библиотеки angular-гридов [right-angled](#) (варианты поинтереснее).

Настраиваем continuous integration с travis ci

Начнем с очевидного — каждому проекту нужен continuous integration. Тут нам на помощь готов прийти [travis ci](#).

Настройка билда в travis достаточно проста и состоит из следующих шагов:

1. Логинимся в travis при помощи своего github-аккаунта.
2. Указываем на странице настроек, какие из наших репозиториев travis должен мониторить.
3. Добавляем в корневой каталог репозитория файл ".travis.yml", в котором указаны настройки окружения и команды для запуска билда.

Например, вот так:

```
language: node_js
node_js:
  - "6"
script:
  - npm run ci
```

Так мы сообщаем travis, что нам необходимо окружение с nodejs версии 6.

Процесс билда состоит из одной npm-команды, которая запускает скрипт с названием "ci", из секции "scripts" файла package.json.

В нашем случае эта команда по очереди выполняет lint проекта с [tslint](#), сборку с [typescript](#), и прогоняет тесты при помощи [karma](#).

Если вам интересны детали, то можно посмотреть их в [package.json на github](#).

Также обратите внимание, что в командах билда мы не прописывали “npm install”. Travis сам понимает, что необходимо выполнить установку зависимостей через npm и выполняет ее.

Более того, если вы используете yarn, то travis поймет это, установит [yarn](#) и выполнит установку зависимостей с помощью него.

В travis заложено множество подобных шаблонов, которые избавляют нас от лишних действий.

Теперь выполняем push файла “.travis.yml”, после чего будет выполнен первый билд. Процесс билда можно наблюдать в реальном времени на странице travis.


Что еще

- Если вас интересуют более сложные сценарии или другие платформы, рекомендуем посмотреть [документацию travis](#). Качество документации впечатляет.
- В случае, если билд закончится ошибкой, travis напишет вам об этом на почту, указанную в аккаунте github. Также можно настроить интеграцию со slack.
- При создании tag, travis сообщит о результате и в случае успешного билда тоже.
- Кроме commit-ов, travis запускает билд на каждый pull request. По завершении билда информация о результате будет отображена на странице pull request-a. После выполнения



merge, travis запустит билд еще раз и опять сообщит о результате на странице pull request-a.


Merged fshchudlo merged 1 commit into `master` from `sinonFakeTimers` on 28 Mar 2016

Conversation 0 Commits 1 Files changed 1

 **nikolaymatrosov** commented on 28 Mar 2016 Collaborator


Rewrote async test synchronously using `sinon.useFakeTimers`.

  Use Sinon fake timers. ✓ c5ccd1b

 **fshchudlo** merged commit `a51f6ad` into `master` on 28 Mar 2016 Hide details Revert

2 checks passed

✓	continuous-integration/travis-ci/pr The Travis CI build passed	Details
✓	continuous-integration/travis-ci/push The Travis CI build passed	Details

 **nikolaymatrosov** deleted the `sinonFakeTimers` branch on 29 Mar 2016 Restore branch

- Вы можете добавить в файл `readme` специальный `badge`, который будет отображать текущий статус билда. По совместительству это ссылка на информацию о последнем билде в `travis`. Используя `badge`, вы можете показать вашему `community`, что с билдом все хорошо. Или все плохо. На второй случай такой `badge` неплохо мотивирует как можно быстрее исправить ситуацию.
- Для публичных репозиторий информация о билдах будет доступна всем желающим. Им даже не нужно будет для этого логиниться в `travis`.

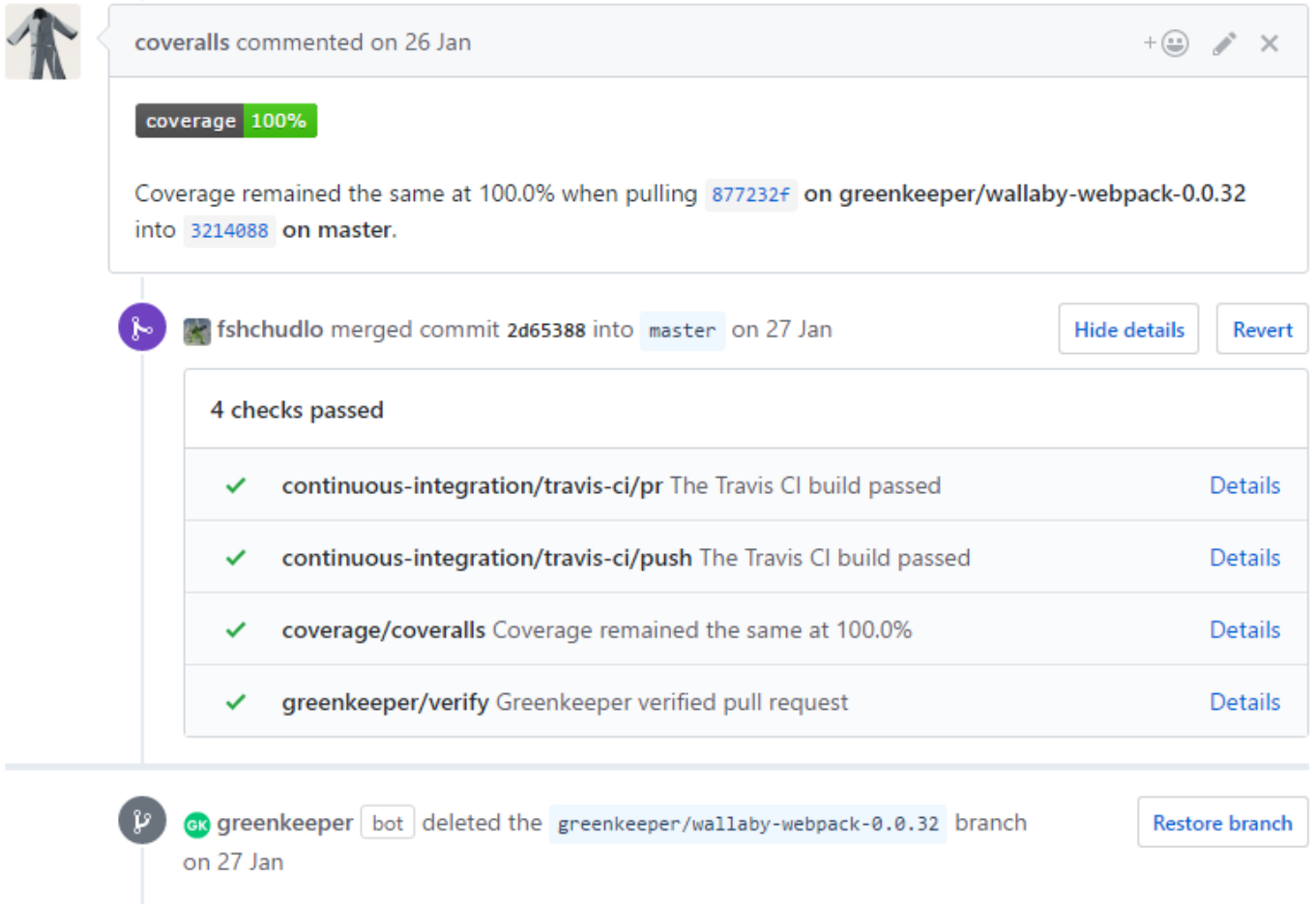
- И последнее. Вы не можете удалять билды из истории. Максимум, что можно сделать, это удалить log билда. Например, если вы “засветили” в нем секретную информацию. Но сам билд останется в истории навсегда.

Настраиваем отчеты о test coverage с coveralls

Следующий сервис, с которым мы познакомимся, это [coveralls](#).

Общая идея состоит в том, что при прогоне тестов вы генерируете отчет о покрытии в формате lcov и отправляете его в сервис coveralls для анализа. Сервис ее обрабатывает и предоставляет следующие возможности:

- Можно добавить badge с процентом покрытия тестами в ваш readme. Так вы сообщите вашему community, что у вас все серьезно.
- При создании pull request-ов coveralls добавляет к pull request-у информацию о том, как изменилось покрытие тестами.



The screenshot displays a GitHub commit history. At the top, a comment from 'coveralls' dated 26 Jan states: 'coverage 100%'. Below this, a commit by 'fshchudlo' dated 27 Jan is shown, which merged commit '2d65388' into the 'master' branch. This commit includes four checks that all passed: 'continuous-integration/travis-ci/pr', 'continuous-integration/travis-ci/push', 'coverage/coveralls', and 'greenkeeper/verify'. The 'coverage/coveralls' check specifically notes that 'Coverage remained the same at 100.0%'. Below the commit list, a message from the 'greenkeeper' bot dated 27 Jan indicates that the 'greenkeeper/wallaby-webpack-0.0.32' branch has been deleted, with a 'Restore branch' button available.

coveralls commented on 26 Jan

coverage 100%

Coverage remained the same at 100.0% when pulling [877232f](#) on greenkeeper/wallaby-webpack-0.0.32 into [3214088](#) on master.

fshchudlo merged commit 2d65388 into master on 27 Jan [Hide details](#) [Revert](#)

4 checks passed

✓	continuous-integration/travis-ci/pr	The Travis CI build passed	Details
✓	continuous-integration/travis-ci/push	The Travis CI build passed	Details
✓	coverage/coveralls	Coverage remained the same at 100.0%	Details
✓	greenkeeper/verify	Greenkeeper verified pull request	Details

greenkeeper bot deleted the greenkeeper/wallaby-webpack-0.0.32 branch on 27 Jan [Restore branch](#)

- Вы можете выставить границы процента покрытия тестами. И, если изменения в pull request-е снижают процент покрытия ниже заданного значения, то проверка pull request-а заканчивается ошибкой.
- Сервис интегрируется с ci-сервисами, такими как travis, хранит историю билдов и позволяет смотреть динамику покрытия тестами от билда к билду. Например, [вот здесь](#) хранятся репорты для библиотеки right-angled.

Чтобы подключить coveralls к вашему проекту, необходимо выполнить следующие шаги:

1. Залогиниться в coveralls при помощи вашего github аккаунта.
2. Выбрать репозитории, для которых вы хотите включить сбор информации.
3. Установить npm-пакет [coveralls](#) в ваш репозиторий.
4. Настроить генерацию coverage reports при прогоне тестов.

Мы не будем рассматривать способы генерации отчетов о покрытии в деталях, поскольку все зависит от того, на чем вы пишете тесты, какие инструменты используете, и как тесты запускаете. Плюс, к примеру, для typescript нет инструментов генерации coverage reports, поскольку запускается на исполнение не typescript, а javascript. И, если вам хочется смотреть покрытие именно typescript кода, то понадобятся инструменты для ремаппинга отчетов о покрытии js-кода обратно на код typescript. Все это излишняя специфика, которой мы стараемся избегать в данной статье. Начать изучение возможных вариантов можно со странички npm-пакета coveralls на github.

5. Добавить npm-скрипт для передачи сгенерированного отчета в сервис coveralls.

```
"scripts": {  
  "coveralls": "cat ./coverage/lcov.info |  
  ./node_modules/.bin/coveralls",  
  ...  
}
```

6. Выполнить добавленный скрипт после прогона билда в travis. Для этого используем секцию `after_success` в файле `travis.yml`. Теперь наш `travis.yml` выглядит следующим образом:

```
language: node_js
node_js:
  - "6"
script:
  - npm run ci
after_success:
  - npm run coveralls
```

Что еще

Помимо github, coveralls интегрируется с bitbucket и обещается скорая поддержка gitlab. Также поддерживается интеграция со множеством сервисов continuous integration и множеством платформ разработки.

Мониторим статус зависимостей с david

Сервис мониторинга зависимостей [david](#) будет самым простым из рассматриваемых в статье.

Идея сервиса проста — мы добавляем на страницу проекта badge, который является индикатором состояния зависимостей проекта. Он же является ссылкой на страничку анализа зависимостей проекта.

Чтобы подключить к нашему проекту david, набираем в браузере адрес по следующему шаблону:

В зависимости от того, какие типы зависимостей есть в вашем проекте, на открывшейся странице вы увидите вкладки “dependencies”, “devdependencies”, “peer dependencies” и “optional dependencies”.

На каждой из вкладок приведен список зависимостей определенного типа и badge, кликнув на который можно скопировать ссылку в формате markdown или HTML и разместить ее у себя в файле readme или на страничке проекта в github pages.

Что еще

Также существует утилита [david cli](#), призванная помочь в обновлении зависимостей на вашей машине. Правда мы так и не смогли понять, в чем ее преимущества по сравнению со встроенными командами [npm outdated](#) и [npm update](#).

Настраиваем автоматическое обновление зависимостей с greenkeeper

Сервис [greenkeeper](#) также призван помочь нам в нелегком деле обновления зависимостей, но делает это на куда более высоком уровне. Его задача — по возможности полностью избавить нас от работы с зависимостями.

Чтобы подключить greenkeeper к нашему проекту, необходимо перейти на страницу greenkeeper в разделе [public integrations](#) на [github](#) и установить приложение в нужный нам репозиторий.

Буквально через минуту greenkeeper создаст pull request, в котором обновит все зависимости вашего проекта и добавит в readme badge, отображающий статус greenkeeper.

Только если вы сделаете merge данного pull request-a, greenkeeper начнет мониторить ваш проект.

Далее, при появлении новых версий зависимостей greenkeeper будет создавать pull request-ы, в которых будет приводить детальное описание, что было обновлено и по возможности приведет список изменений в новой версии. Вам остается только сделать merge.



greenkeeper bot commented on 26 Jan



Version 0.0.32 of wallaby-webpack just got published.

Dependency	wallaby-webpack
Current Version	0.0.31
Type	devDependency

The version 0.0.32 is not covered by your current version range.

Without accepting this pull request your project will work just like it did before. There might be a bunch of new features, fixes and perf improvements that the maintainers worked on for you though.

I recommend you look into these changes and try to get onto the latest version of wallaby-webpack. Given that you have a decent test suite, a passing build is a strong indicator that you can take advantage of these changes by merging the proposed change into your project. Otherwise this branch is a great starting point for you to work on the update.

Commits

The new version differs by 2 commits .

- a2f948b Version bump to 0.0.32
- 4e00eef Added preserveEntryFileLoadOrder setting to load entry files respecting how the first of them is positioned in the files list

See the [full diff](#).

► Not sure how things should work exactly?

Your Greenkeeper Bot 🌿

Reviewers

No reviews—request one

Assignees

No one—assign yourself

Labels

greenkeeper

Projects

None yet

Milestone

No milestone

Notifications

Unsubscribe

You're receiving notifications because you modified the open/close state.

3 participants



Lock conversation

Естественно, подключение greenkeeper имеет смысл только если у вас настроен автоматический билд и имеются тесты, при помощи которых можно хотя бы номинально проверить, что ваш проект находится в рабочем состоянии после обновления. Greenkeeper распознает, есть ли в вашем репозитории билд и тесты, и пишет предупреждение в тексте pull request-a, если не обнаруживает того или другого.

Что еще

- После того, как вы закроете pull request, greenkeeper сам удалит созданный branch.
- Если у обновленной зависимости очень быстро выходят следующие версии (что будет похоже на выпуск quick fix-a), greenkeeper создаст новый branch и напишет об этом в комментариях к исходному pull request-у.
- Вы можете заметить, что иногда greenkeeper создает branch, и сразу его удаляет, не делая pull request. Так greenkeeper действует если вышло обновление зависимости, которые вписывается в указанный у вас в package.json [version range](#). То есть данная версия зависимости и так попадет к вам при переустановке зависимостей. Greenkeeper же таким образом проверяет, что даже попадающая в version range версия не ломает ваш билд и после его прогона просто удаляет branch. В случае ошибки greenkeeper создаст issue.
- Если вы не хотите, чтобы greenkeeper обновлял определенные зависимости в вашем проекте, то вы можете добавить в ваш файл package.json следующую секцию:

```
"greenkeeper": {  
  "ignore": ["список", "зависимостей", "которые", "не нужно",  
    "обновлять"]  
}
```

- Также вам стоит знать, что на текущий момент greenkeeper не работает с yarn. Есть способы обойти эту проблему, но назвать их хорошими трудно. Вся информацию по данному вопросу можно посмотреть в [issue на github](#).
- И последний момент. Возможна ситуация, когда в проекте используется несколько зависимостей, которые необходимо обновлять одновременно. Например, [angular](#) разбит на модули,

и велики шансы, что в проекте понадобится больше одного модуля. При этом, обновлять их имеет смысл все вместе. Greenkeeper пока не в состоянии справиться с такой задачей.

Улучшаем сообщения коммитов с commitizen

[Commitizen](#) это целый набор инструментов, помогающих в деле написания содержательных сообщений к коммитам.

Помимо большей информативности для людей, заинтересованных в вашем репозитории, использование commitizen имеет еще один плюс. Из генерируемых commitizen сообщений можно легко собирать changelog-и и release notes. И есть даже утилиты, анализирующие историю коммитов и подсказывающие, какой должен быть следующий номер версии, чтобы соответствовать конвенции [semver](#). Но об этом в следующем разделе.

А начнем мы с самого простого. Установим [cz-cli](#). После установки мы можем использовать команду “git cz” вместо “git commit”. Данная команда запускает визард, который проводит нас через серию вопросов о том, что именно мы изменили в текущем коммите и генерирует сообщение коммита [в формате](#), который одновременно легко читается человеком и парсится различными инструментами.

Итак:

1. Устанавливаем commitizen глобально:

```
npm install commitizen -g
```

2. Делаем наш репозиторий commitizen friendly. Для этого в папке репозитория запускаем команду:

```
commitizen init <название адаптера> --save-dev --save-exact
```

Вместо <название адаптера> необходимо указать [один из возможных адаптеров](#). В мире front end разработки наиболее популярным является cz-conventional-changelog, следующий [нотации сообщений, разработанной командой angular](#)

Данная команда установит необходимые зависимости, сохранит их в секции devDependencies файла package.json и там же пропишет необходимые настройки.

Что еще

- Помимо работы через командную строку, существуют расширения для редакторов кода. Например, для [vscode](#). Возможно с ним вам будет удобнее, хотя принципиальных отличий от использования “git cz” в терминале того же vscode нет.
- Стоит заметить, что сообщения, которые генерирует git cz, несложно писать и самому. Некоторые из наших разработчиков так и делают. Полное описание формата можно [посмотреть здесь](#).
- Вы можете [добавить badge](#) на страницу readme, чтобы посетители вашего репозитория знали, что вы следуете

конвенции commitizen.

Генерируем changelog и release notes с conventional-changelog

После того, как мы подключили commitizen, мы получаем еще одну опцию — возможность настроить автоматическую генерацию changelog-а и release notes.

Поможет нам в этом [conventional-changelog](#).

conventional-changelog это целое семейство инструментов, при помощи которых можно построить выпуск релиза как при помощи высокоуровневых инструментов по готовым шаблонам, так и собрать из отдельных инструментов то, что нужно конкретно вам.

Сами разработчики рекомендуют использовать [standard-version](#), который является относительно высокоуровневым инструментом.

Еще более высокоуровневым является [semantic-release](#). Данный инструмент даже сам делает push изменений и публикует версию в npm.

Из нашего опыта — попробовав оба варианта, мы остановились на использовании более низкоуровневых инструментов.

Причина такого выбора в том, что standard-version, например, не делает push изменений и не генерирует описание релиза на github.

Добавление к процессу релиза таких вещей плюс настройка через опции `standard-version` запуска билда, или генерации документации вкпе требуют настройки, которая по сложности сопоставима с ручной настройкой процесса публикации на основе команды `npm version`.

`Semantic-release`, с другой стороны, пытается автоматизировать процесс целиком и для его использования необходимо очень дисциплинированно подходить к разработке. Сложность его настройки также сопоставима с ручной настройкой процесса. И последнее. `Semantic-release` делает `npm publish`, что ограничивает его использование только для библиотек, распространяемых через `npm`, а публиковать версии можно не только для библиотек.

Итак, мы построим процесс релиза на основе `npm version`. Также при запуске скрипта `version` `npm` самостоятельно запускает скрипты `preversion` и `postversion`, если таковые имеются. Мы будем использовать оба. Процесс выпуска релиза будет состоять из следующих шагов:

Фаза `preversion`:

1. Очистка директорий с результатами предыдущей сборки при помощи `rimraf`.
2. Прогон `tslint`.
3. Компиляция `typescript`.
4. Прогон тестов с `karma`.

На примере репозитория `e2e4`, данные шаги идентичны

выполняемым в хуке `precommit`, поэтому мы используем тот же скрипт:

```
{
  "preversion": "npm run precommit",
  "precommit": "npm run rimraf -- esm coverage && npm run clean:src &&
  npm run clean:tests && npm run lint && npm run build && npm run
  test"
}
```

Фаза `version`:

1. Сгенерировать документацию для публикации на `github pages`. В нашем случае для этого используется [typedoc](#). Касательно `github pages` — с недавних времен стало не обязательно создавать `branch` с именем `gh-pages`, можно просто [указать в настройках github](#) папку вашего проекта, которая будет использоваться в качестве сайта. По умолчанию это папка `docs`. Поэтому мы просто генерируем документацию в папке `docs`.
2. Добавить сгенерированную документацию в `git` для коммита.
3. Дополнить `changelog.md` информацией об изменениях. Для этого мы будем использовать [conventional-changelog-cli](#). В качестве параметров передаем ему название файла и конвенцию для разбора сообщений. В нашем случае это `angular`.
4. Добавить обновленный `changelog` для коммита.
5. Обновить версию в `package.json` и добавить его для коммита. Это `npm` сделает за нас.

6. Сгенерировать tag для версии и поставить на него метку pre-release или latest, если необходимо. Это npm также сделает за нас. Какую метку поставить он разберется по номеру версии, в соответствии с правилами [semver](#).

Итого, получается следующий набор скриптов:

```
{
  "version": "npm run docs && git add -A docs && npm run changelog && git add CHANGELOG.md",
  "changelog": "npm run conventional-changelog -- -p angular -i CHANGELOG.md -s",
  "docs": "npm run rimraf -- docs && typedoc --options typedoc.json src/"
}
```

Фаза postversion:

1. Выполняем push измененных package.json, changelog.md, документации.
2. Выполняем push созданного tag.
3. Дописываем в github release описание изменений. Для этого используем conventional-github-releaser, передавая ему в качестве параметра нотацию сообщений. В нашем случае это “angular”. Поскольку для записи информации необходимы соответствующие права доступа, необходимо сгенерировать access token, который conventional-github-releaser будет использовать для авторизации. Как это делается можно посмотреть в описании проекта.

Скрипт для version:

```
{  
  "postversion": "git push && git push --tags && conventional-github-releaser -p angular",  
}
```

Наш процесс выпуска релиза готов.

Запускаем скрипт командой:

```
npm version <номер версии или major/minor/patch>
```

Сгенерированные release notes выглядят примерно так:

Pre-release

v2.0.0-beta.9

ee615b9

v2.0.0-beta.9

fshchudlo released this on 3 Dec 2016 · 60 commits to master since this release

Code Refactoring

- progress state: `ProgressState` renamed to `OperationStatus`, `StateTrackingService` renamed to `StatusTrackingService` (9b1bd73)

Features

- async subscriptions: `AsyncSubscriber` class added to manage subscriptions in abstract manner (738e6cb)
- lists: `List` service added (811d236)
- state management: abstract `StateService` base class added (7b65b8d)

BREAKING CHANGES

- progress state: `ProgressState` renamed to `OperationStatus`, `StateTrackingService` renamed to `StatusTrackingService`

Downloads

[Source code \(zip\)](#)

[Source code \(tar.gz\)](#)

Содержимое в `changelog.md` почти такое же, поэтому мы не будем его приводить.

Что еще

- Использовать инструменты `conventional-changelog` можно не только с `github`. `Conventional-changelog-cli` работает с любым `git`-репозиторию, а у `conventional-github-releaser` есть [братская библиотека для работы с gitlab](#).
- Возможно, вы задались вопросом, зачем размещать одну и ту же информацию в двух местах. Описание релиза удобно, когда пользователь просто заинтересован, что в новой версии. А `changelog` поставляется в `npm`-пакете и пригодится когда пользователь уже обновил версию и ему нужна информация о релизе, чтобы внести коррективы в код в соответствии с

breaking changes или использовать новые возможности библиотеки. И смотреть данную информацию прямо в редакторе кода удобнее, чем в описании релиза на github.

Управляем задачами с zube

О сервисе zube.io мы не будем рассказывать много. Каждый из нас работал с таск-трекерами и zube мало чем отличается от множества из них.

Назовем лишь два его достоинства, из-за которых мы решили упомянуть его в данной статье:

1. Он бесплатный для открытых проектов, что редкость для инструментов управления проектами, интегрирующихся с github.
2. zube позволяет объединять задачи из нескольких github репозиторий в один проект и работать с ними одновременно.

На этом мы заканчиваем наш обзор. Если вам известны другие интересные сервисы или инструменты для работы с github — пожалуйста, поделитесь своим знанием в комментариях.

Спасибо за уделенное внимание.

Изображение обложки для статьи — [the Benevocats](#) by cameronmcefee.



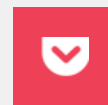
+39



Поделиться:



Сохранить:



Комментарии (16)

Похожие публикации

Безопасность мобильных приложений, или «Кто проверит проверяющих?»

eastbanctech • 4 февраля 2015 в 11:39

8

Полезный Open Source и как мы научили Zxing разговаривать на другом языке

eastbanctech • 7 марта 2014 в 12:21

0

Как использовать функцию обработки входящей почты в SharePoint 2010 — пример из практики

eastbanctech • 6 февраля 2014 в 15:54

1

Популярное за сутки

Яндекс открывает Алису для всех разработчиков. Платформа Яндекс.Диалоги (бета)

BarakAdama • вчера в 10:52

69

Почему следует игнорировать истории основателей успешных стартапов

ПЕРЕВОД

m1rko • вчера в 10:44

20

Как получить телефон (почти) любой красоты в Москве, или интересная особенность MT_FREE

ИЗ ПЕСОЧНИЦЫ

sab404 • вчера в 20:27

24

Java и Project Reactor

zealot_and_frenzy • вчера в 10:56

10

Пользовательские агрегатные и оконные функции в PostgreSQL и Oracle

erogov • вчера в 12:46

6

Лучшее на Geektimes

Как фермеры Дикого Запада организовали телефонную сеть на колючей проволоке

NAGru • вчера в 10:10

31

Энтузиаст сделал новую материнскую плату для ThinkPad X200s

49

alizar • вчера в 15:32

Кто-то посылает секс-игрушки с Amazon незнакомцам. Amazon не знает, как их остановить

85

Pochtoycom • вчера в 13:06

Илон Маск продолжает убеждать в необходимости создания колонии людей на Марсе

139

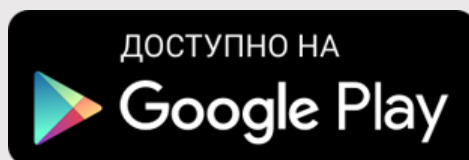
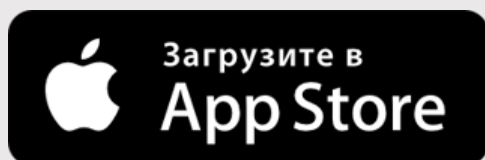
marks • вчера в 14:19

Дела шпионские (часть 1)

16

TashaFridrih • вчера в 13:16

Мобильное приложение



Полная версия

2006 – 2018 © TM