



# РАБОЧЕЕ ОКРУЖЕНИЕ



АЛЕКСАНДР ШЛЕЙКО



# АЛЕКСАНДР ШЛЕЙКО

Программист в Яндекс



[a.shleyko@yandex.ru](mailto:a.shleyko@yandex.ru)



[vk.com/shleiko](https://vk.com/shleiko)



[@dustyo\\_0](https://t.me/@dustyo_0)



# ПЛАН ЗАНЯТИЯ

1. [Пакетные менеджеры: npm, yarn](#)
2. [ESLint, Babel, Webpack, Webpack DevServer, Jest](#)
3. [Continuous Deployment](#)
4. [GitHub Pages](#)



# FRONTEND

Современная фронтенд-разработка включает в себя достаточно этапов, напрямую не относящихся к созданию кода:

- организация проекта;
- шаблонизация типовых операций;
- настройка окружения (http-server);
- управление зависимостями;
- тестирование и сборка;
- deployment.



# ЭФФЕКТИВНОСТЬ И АВТОМАТИЗАЦИЯ

В основе организации всего процесса разработки стоит эффективность — насколько быстро и удобно мы можем выполнять те или иные действия.

Насколько максимально мы можем автоматизировать повторяющиеся и рутинные операции, чтобы тратить время на другие, более полезные активности.



# МЕНЕДЖЕРЫ ПАКЕТОВ

# NPM, YARN

Пакетные менеджеры, предназначенные для:

- организации проекта;
- шаблонизации типовых операций;
- управления зависимостями.

Стандартом де-факто является `npm`. `yarn` является также достаточно частым выбором, используется в ряде популярных проектов: `Webpack`, `React`, `Jest` и других.

---

# ЗАДАЧА

Вы достаточно хорошо овладели `npm` (должны были, по крайней мере). Но в вашей новой команде используется другой пакетный менеджер — `yarn`.

Нужно на базовом уровне овладеть его использованием, чтобы свободно себя чувствовать вне зависимости от используемого пакетного менеджера.



# УСТАНОВКА

`npm` уже входит в состав дистрибутива Node.js. `yarn` же можно установить с помощью `npm`:

```
npm install --global yarn  
yarn --version
```

# ИНИЦИАЛИЗАЦИЯ

```
npm init
```

```
yarn init
```

Результатом выполнения данной команды станет файл package.json, содержащий мета-данные вашего пакета.

---

# GIT

Если в каталоге, где вы инициализируете проект, уже существует Git-репозиторий, то в `package.json` автоматически пропишется секция `repository`:

```
"repository": {  
  "url": "https://github.com/coursar/demo",  
  "type": "git"  
},
```

# УСТАНОВКА ЗАВИСИМОСТЕЙ

Для обычных зависимостей (необходимых для функционирования приложения):

```
npm install <dependency-name>
```

```
yarn add <dependency-name>
```

В результате установки зависимостей `npm` создаёт файл package-lock.json, описывающий всё дерево зависимостей. Для `yarn` же роль подобного файла играет yarn.lock.

Обратите внимание, и package-lock.json, и yarn.lock рекомендуется хранить в репозитории (в рамках курса — обязательно).

# УСТАНОВКА DEV-ЗАВИСИМОСТЕЙ

```
npm install --save-dev <dependency-name>
```

```
yarn add --dev <dependency-name>
```

**Вопрос:** в какую секцию записываются dev-зависимости?

# УДАЛЕНИЕ ЗАВИСИМОСТЕЙ

```
npm uninstall <dependency-name>
```

```
yarn remove <dependency-name>
```

# СКРИПТЫ

Скрипты прописываются в секции `scripts` файла `package.json`. При этом для запуска скриптов:

- `npm`
  - `npm run <script-name>`
  - `npm <script-name>` (если `<script-name>` одно из стандартных имён)
- `yarn`
  - `yarn run <script-name>`
  - `yarn <script-name>` (если `<script-name>` не одно из зарезервированных имён)

Обратите внимание на различие во вторых пунктах.

# СКРИПТЫ

Кроме того, для передачи аргументов в скрипт:

```
npm run <script-name> -- <options>
```

```
yarn run <script-name> <options>
```



---

# СКРИПТЫ

В состав `npm` входит утилита `npx`, которая позволяет запускать исполняемые файлы из `node_modules/.bin`.

Для `yarn` аналогичной возможностью обладает `yarn run`.

`yarn run` умеет запускать только установленные исполняемые файлы или скрипты.



## УСТАНОВКА ЗАВИСИМОСТЕЙ ИЗ PACKAGE.JSON

```
npm install
```

```
yarn
```

# КАКОЙ МЕНЕДЖЕР ИСПОЛЬЗОВАТЬ?

Какой менеджер лучше, быстрее, удобнее и т.д.?

Вы должны уметь использовать оба. В остальном — всё зависит от предпочтений.

# КАКОЙ МЕНЕДЖЕР ИСПОЛЬЗОВАТЬ?

Какой менеджер использовать на курсе?

Если в задаче явно не указан необходимый менеджер — мы оставляем выбор на ваше усмотрение.



# **ИНСТРУМЕНТЫ РАЗРАБОТКИ**



# ИНСТРУМЕНТЫ РАЗРАБОТКИ

Стандартный набор на курс включает в себя:

- Babel;
- Jest;
- ESLint;
- Webpack;
- Webpack DevServer.

**Есть ли у вас какие-либо проблемы с установкой/  
настройкой этих инструментов?**



# ПЛАГИНЫ И ЛОАДЕРЫ WEBRACK

Ознакомьтесь самостоятельно со списком и функциональностью самых популярных плагинов и лоадеров:

- [плагины](#);
- [лоадеры](#).



# РАЗДЕЛЕНИЕ КОНФИГУРАЦИЙ

Достаточно часто конфигурацию Webpack разделяют на несколько частей, поскольку часть плагинов нужна только для Production-режима (например, плагины, которые сжимают изображения)

Поэтому, как минимум, выделяют следующие части:

- `prod` - настройки для режима `prod`
- `dev` - настройки для режима `dev` (Dev-сервер и т.д.)
- `common` - общая часть для `prod` и `dev`

# WEBPACK-MERGE

Для разделения конфигураций нам понадобится специальный пакет webpack-merge

```
npm install --save-dev webpack-merge
```

```
yarn add --dev webpack-merge
```

# WEBPACK-MERGE

Создадим три файла:

- webpack.common.js
- webpack.prod.js
- webpack.dev.js

И перепишем скрипты запуска ( `scripts` файл `package.json` ):

```
"start": "webpack-dev-server --config webpack.dev.js",  
"build": "webpack --config webpack.prod.js",
```



## WEBPACK.COMMON.JS

Файл `webpack.common.js` будет содержать всё, что до этого было в файле `webpack.config.js`

В этом файле будет храниться конфигурация, общая для `prod` и `dev`.

# WEBPACK.PROD.JS

Именно здесь мы будем добавлять конфигурацию плагинов оптимизации:

```
1  const merge = require('webpack-merge');
2  const common = require('./webpack.common');
3  const TerserPlugin = require('terser-webpack-plugin');
4  const OptimizeCSSAssetsPlugin = require('optimize-css-assets-webpack-plugin');
5  module.exports = merge(common, {
6    mode: 'production',
7    optimization: {
8      minimizer: [
9        new TerserPlugin({}),
10       new OptimizeCSSAssetsPlugin({}),
11     ],
12   },
13 });
```

При этом нужно не забыть установить `terser-webpack-plugin*`, `optimize-css-assets-webpack-plugin`.

Примечание:\* `terser-webpack-plugin` идёт в составе Webpack, но лучше явно фиксировать зависимости в `package.json`.

# WEBPACK.DEV.JS

```
1  const merge = require('webpack-merge');  
2  const common = require('./webpack.common');  
3  module.exports = merge(common, {  
4    mode: 'development',  
5    devtool: 'inline-source-map',  
6  });
```



# CONTINUOUS DEPLOYMENT



# ТЕСТИРОВАНИЕ, СБОРКА И DEPLOYMENT

Одна из самых трудоёмких (по времени) и часто повторяющихся задач — это цепочка, состоящая из:

1. Прогон тестов.
2. Создания сборки.
3. Разворачивания сборки на сервере.

Если автоматизировать этот процесс, то мы получим значительную экономию времени.





# ЗАДАЧА

Максимальная автоматизация процесса тестирования, сборки и развёртывания.

---

# CONTINUOUS DEPLOYMENT



*Continuous Deployment means that every change goes through the pipeline and automatically gets put into production, resulting in many production deployments every day.*

***Martin Fowler***



*Перевод:*

*Continuous Deployment означает, что любое изменение проходит через конвейер (pipeline) и автоматически разворачивается на production (боевые сервера), позволяя осуществлять большое количество развёртываний каждый день.*

# CONTINUOUS DEPLOYMENT



*Continuous deployment can be thought of as an extension of continuous integration, aiming at minimizing lead time, the time elapsed between development writing one new line of code and this new code being used by live users, in production.*

*To achieve continuous deployment, the team relies on infrastructure that automates and instruments the various steps leading up to deployment, so that after each integration successfully meeting these release criteria, the live application is updated with new code.*

***AgileAlliance.org***



*Перевод:*

*Continuous deployment можно рассматривать как расширение continuous integration, нацеленное на минимизацию времени выхода на рынок, времени между написанием кода и предоставлением этого кода живым пользователям в production-среде.*

*Для достижения continuous deployment, команда полагается на инфраструктуру, которая автоматизирует шаги для развёртывания, поэтому после каждой интеграции, удовлетворяющей критериям релиза, приложение обновляется до новой версии.*



# CONTINUOUS DEPLOYMENT ДЛЯ НАС

Мы не будем вдаваться в тонкости формулировок, обозначим для себя следующие ключевые моменты:

1. Выстраивание такой инфраструктуры, при которой любое изменение кода будет проходить серию автоматизированных проверок в специальной среде.
2. В случае, если все проверки будут пройдены успешно, та же среда развернёт наше приложение на “боевом сервере”.

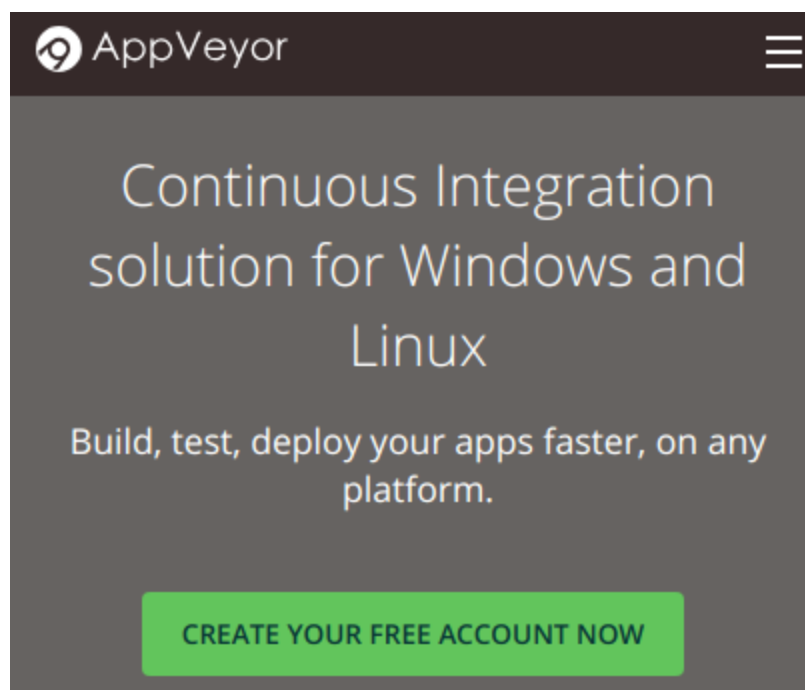
# CONTINUOUS DEPLOYMENT ДЛЯ НАС

- “любое изменение кода” — `git push`;
- проверки — lint’инг и авто-тесты;
- сборка — `webpack build`;
- развёртывание — GitHub Pages;

Плюс видимое в любой момент состояние проекта (собирается или нет).

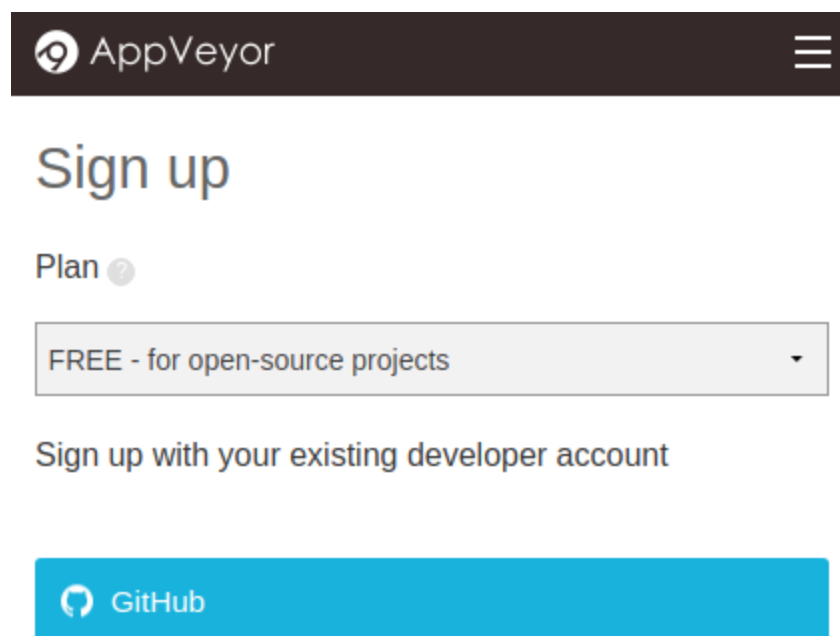
# APPVEYOR

[AppVeyor](#) — одна из платформ, предоставляющих функциональность Continuous Deployment. В базовом варианте бесплатна.



# APPVEYOR

Бесплатный тарифный план для публичных репозиторий GitHub (авторизация — также через GitHub):



The image shows the AppVeyor sign-up page. At the top is a dark header with the AppVeyor logo and a menu icon. Below the header, the text 'Sign up' is displayed. Underneath, there is a 'Plan' label with a help icon, followed by a dropdown menu showing 'FREE - for open-source projects'. Below the dropdown, the text 'Sign up with your existing developer account' is visible. At the bottom, there is a blue button with the GitHub logo and the text 'GitHub'.

AppVeyor

## Sign up

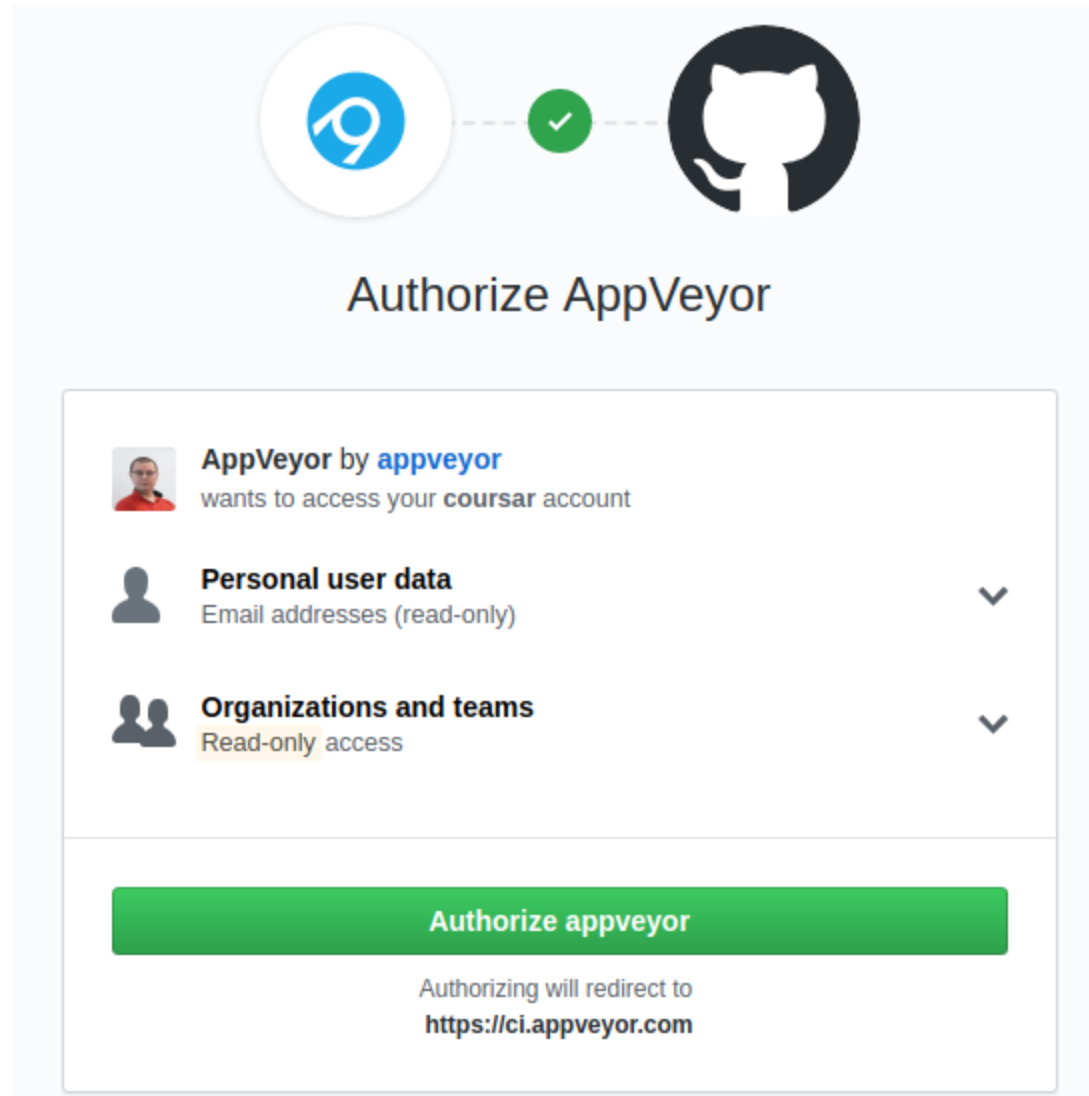
Plan ?

FREE - for open-source projects ▼

Sign up with your existing developer account

GitHub

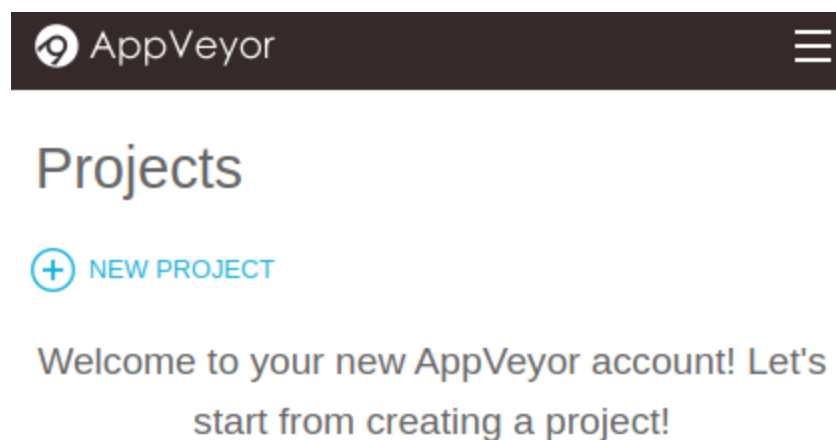
# APPVEYOR & GITHUB





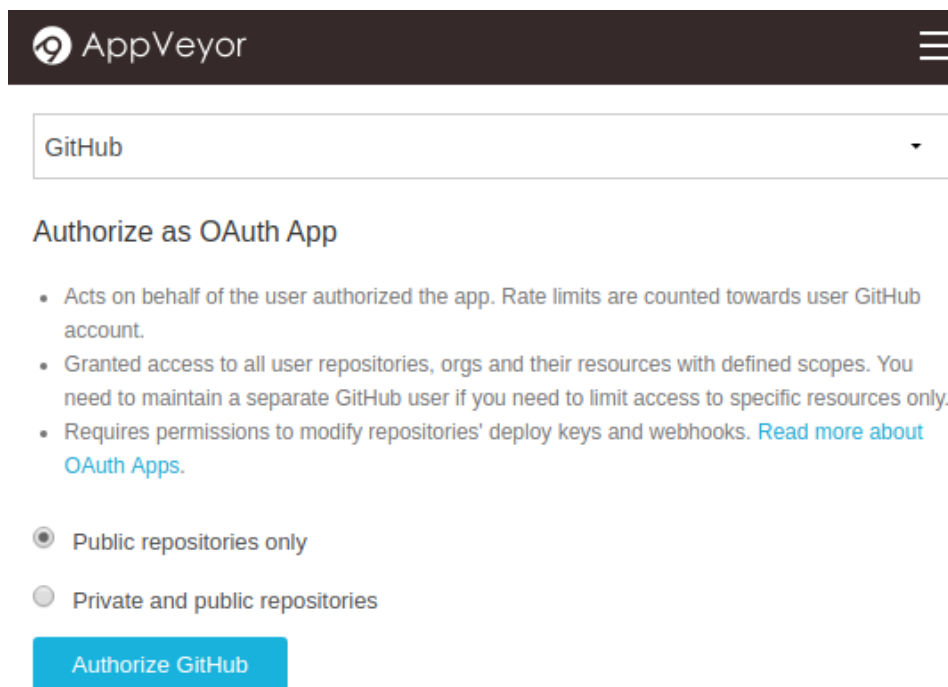
# APRVEYOR. СОЗДАНИЕ ПРОЕКТА

После авторизации станет доступной панель управления, где можно создать новый проект:



# APRVEYOR. СОЗДАНИЕ ПРОЕКТА

Авторизуйте AppVeyor в качестве OAuth App.



The screenshot shows the AppVeyor web interface. At the top is a dark header with the AppVeyor logo and a hamburger menu icon. Below the header is a white box containing a dropdown menu with 'GitHub' selected. Underneath is the heading 'Authorize as OAuth App'. This is followed by a bulleted list of permissions: 'Acts on behalf of the user authorized the app. Rate limits are counted towards user GitHub account.', 'Granted access to all user repositories, orgs and their resources with defined scopes. You need to maintain a separate GitHub user if you need to limit access to specific resources only.', and 'Requires permissions to modify repositories' deploy keys and webhooks. [Read more about OAuth Apps.](#)'. Below the list are two radio button options: 'Public repositories only' (which is selected) and 'Private and public repositories'. At the bottom is a blue button labeled 'Authorize GitHub'.

AppVeyor

GitHub

Authorize as OAuth App

- Acts on behalf of the user authorized the app. Rate limits are counted towards user GitHub account.
- Granted access to all user repositories, orgs and their resources with defined scopes. You need to maintain a separate GitHub user if you need to limit access to specific resources only.
- Requires permissions to modify repositories' deploy keys and webhooks. [Read more about OAuth Apps.](#)

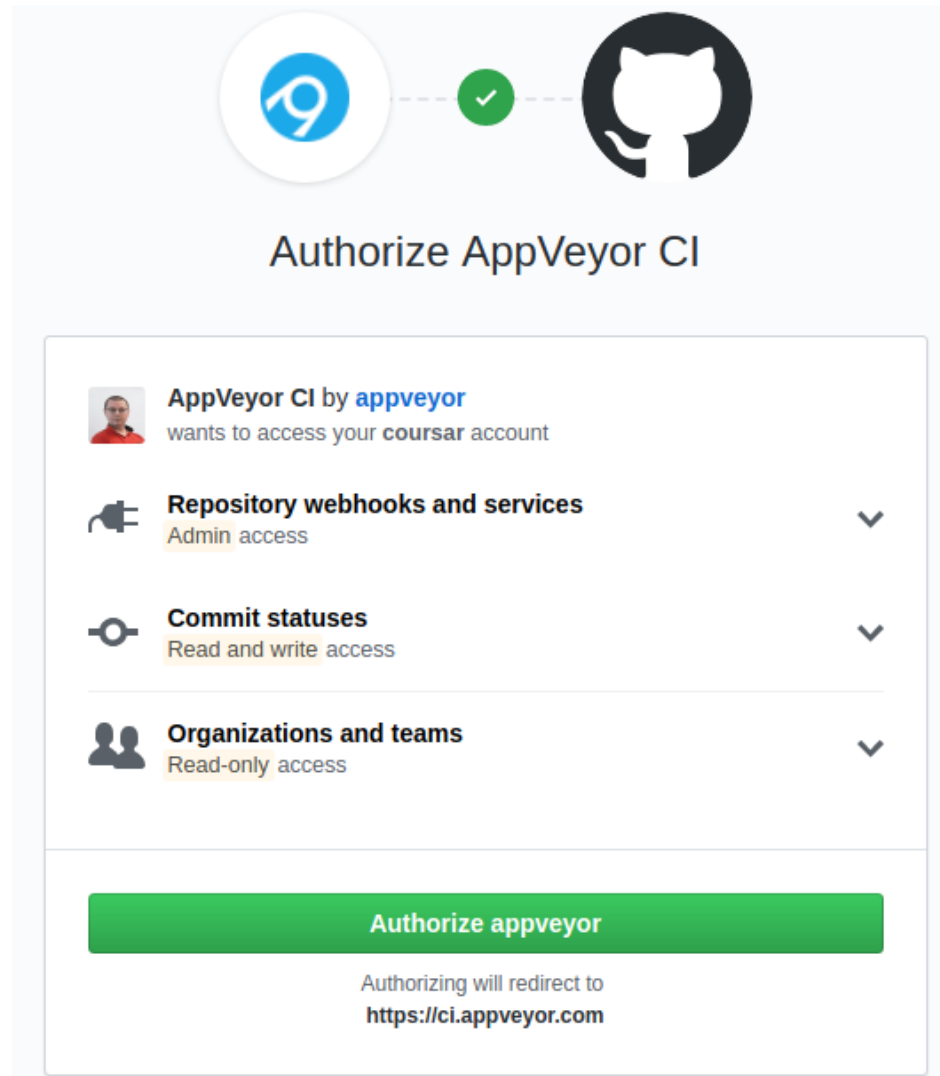
☒ Public repositories only

☐ Private and public repositories

Authorize GitHub

Это даст возможность приложению получать уведомления о ваших push в репозиторий, модификации и т.д.

# APPVEYOR. OAUTH APP





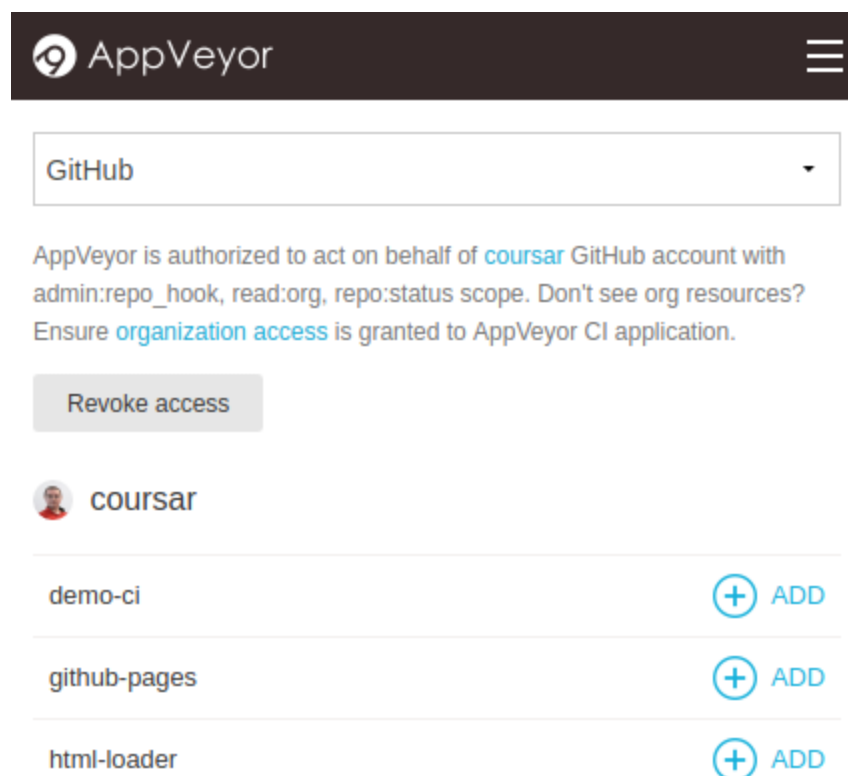
# APPVEYOR. OAUTH APP

Детальнее об OAuth вы можете прочитать на:

- <https://oauth.net/2/>
- <https://auth0.com/docs/protocols/oauth2>

# APRVEYOR. ВЫБОР РЕПОЗИТОРИЯ

После авторизации достаточно будет нажать кнопку **ADD** напротив необходимого репозитория:



## GITHUB. GITHUB\_TOKEN

Для программного взаимодействия с GitHub существует возможность генерации токенов (вместо указания логина и пароля). Токены позволяют относительно безопасно их использовать на сторонних сервисах с возможностью отзыва (и без компрометации основного пароля аккаунта).

Перейдите по адресу <https://github.com/settings/tokens> и нажмите на кнопку `Generate New Token`.

# GITHUB. GITHUB\_TOKEN

Выдайте только `scope` - `repo`:

## Token description

appveyor

What's this token for?

## Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes.](#)

- |   |                                      |
|---|--------------------------------------|
| <input checked="" type="checkbox"/> <b>repo</b>     | Full control of private repositories |
| <input checked="" type="checkbox"/> repo:status     | Access commit status                 |
| <input checked="" type="checkbox"/> repo_deployment | Access deployment status             |
| <input checked="" type="checkbox"/> public_repo     | Access public repositories           |
| <input checked="" type="checkbox"/> repo:invite     | Access repository invitations        |





# APPEYOR.GITHUB\_TOKEN

Хранить токен мы будем в переменных окружения. Для этого необходимо для конкретного проекта зайти на вкладку **Settings** → **Environment** и прописать переменную окружения (**Environment Variables**) **GITHUB\_TOKEN** с тем значением, что вы получили на предыдущем шаге:

The screenshot shows the Appveyor Settings page for a project. The 'Settings' tab is selected in the top navigation bar. On the left sidebar, the 'Environment' tab is highlighted. The main content area shows the 'Environment variables' section. Under 'Build worker image', 'Ubuntu' is selected in a dropdown menu. Below this is a text input field for 'Clone directory' with the placeholder text 'Optional, e.g. c:\projects\myproject'. The 'Environment variables' section has a header with a help icon. Below it, there is a table with one row: the variable name 'GITHUB\_TOKEN' and its value, which is represented by a series of dots indicating a masked secret. An 'Add variable' button is located at the bottom of the table.

Environment variables ?	
GITHUB_TOKEN	.....

# КОНФИГУРАЦИЯ КАК КОД

Поскольку вручную настраивать каждый проект в системе Continuous Deployment — лишняя трата времени, мы будем хранить всю конфигурацию для AppVeyor в специальном файле с названием .appveyor.yml

Файл этот должен храниться в самом репозитории на GitHub, тогда AppVeyor будет автоматически подхватывать настройки из него:

 `.appveyor.yml` `.gitignore` `README.md` `package-lock.json` `package.json`



# YAML

Формат сериализации данных, используемый многими системами для хранения конфигурации.

Ссылки:

- [Wikipedia](#)
- [Спецификация](#)

Странички на Wikipedia достаточно для понимания базовых конструкций языка.



## CMD, PS, BASH

AppVeyor позволяет использовать все три оболочки, но мы для простоты на Linux будем использовать Bash.

# LINUX CONFIG

Файл `.appveyor.yml`

```
image: Ubuntu1804 # образ для сборки
stack: node 10 # окружение
branches:
  only:
    - master # ветка git
cache: node_modules # кеширование
install:
  - npm install # команда установки зависимостей
build: off
build_script:
  - npm run build # команда сборки
test_script:
  - npm run lint && npm test # скрипт тестирования
deploy_script: # скрипт развёртывания
  - git config --global credential.helper store
  - git config --global user.name AppVeyor
  - git config --global user.email ci@appveyor.com
  - echo "https://$GITHUB_TOKEN:x-oauth-basic@github.com" > "$HOME/.git-credentials"
  - npx push-dir --dir=dist --branch=gh-pages --force --verbose
```

# .GIT-CREDENTIALS

Для передачи данных по https git требует указание аутентификационных данных (в нашем случае токена).

Поскольку наша цель — построить автоматизированную систему, необходимо исключить ручной ввод токена.

Git предлагает для этих целей механизм, позволяющий сохранять данные в файле .git-credentials. Git достаточно щепетильно относится к символам переноса строки, используемым в этом файле.

Никогда не храните на GitHub учётные данные в открытом виде - используйте переменные окружения или зашифрованные значения.

## ОПАСНОСТИ NPM

В указанных выше примерах использовался специальный пакет `push-dir`, который позволяет сделать `git push` содержимого указанного каталога (в нашем случае `dist`) в определённый branch (в нашем случае `gh-pages`).

Мы использовали данный пакет только в учебных целях для упрощения процесса. Будьте внимательны, используя сторонние пакеты при deployment'e — они вполне могут “сливать” ваши учётные данные.



# GITHUB PAGES



# GITHUB PAGES

После настройки всего процесса каждый `push` в ветку `master` GitHub-репозитория будет приводить к запуску сборки и deployment'a на AppVeyor.

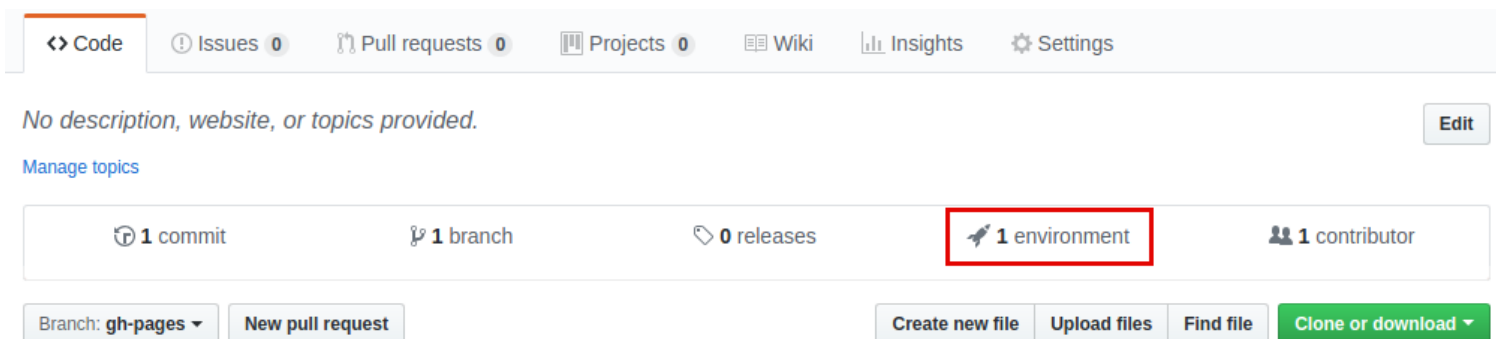
Увидеть развёртывания можно будет на вкладке Environment на странице проекта.

# ENVIRONMENT

GitHub Pages создаст веб-сайт по адресу:

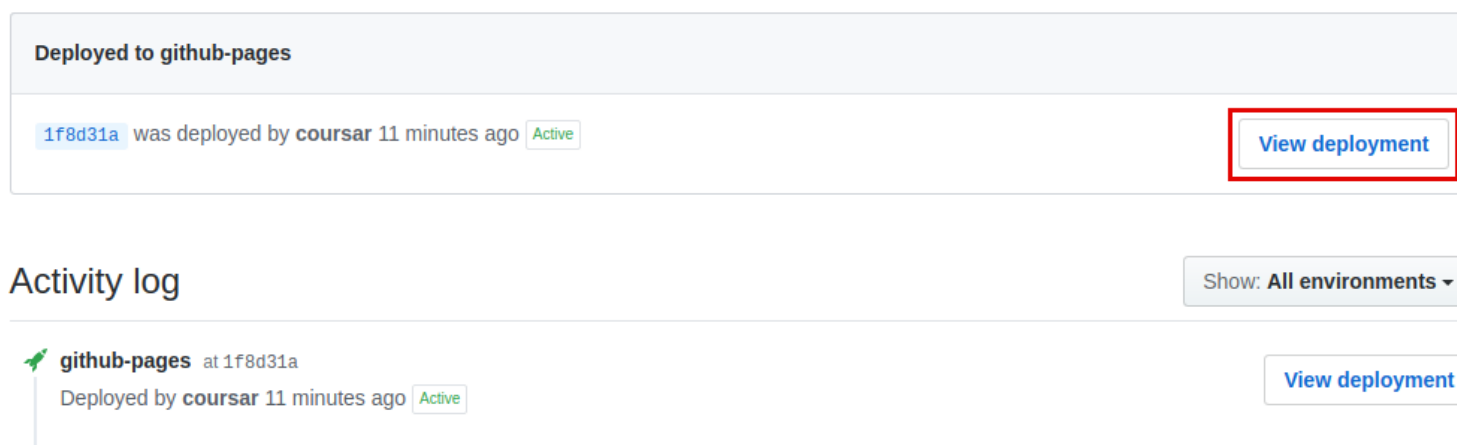
`https://<ваш логин>.github.io/<название репозитория>/`

В интерфейсе можно увидеть на вкладке "Environment" репозитория:



# DEPLOYMENT

На странице будет указана ссылка на сам веб-сайт и история развёртываний:




The screenshot shows a GitHub deployment interface. At the top, a light blue bar indicates 'Deployed to github-pages'. Below this, a deployment entry for commit '1f8d31a' is shown, deployed by 'coursar' 11 minutes ago, with an 'Active' status. A red rectangular box highlights the 'View deployment' link. Below this section is the 'Activity log' header, with a 'Show: All environments' dropdown on the right. The activity log shows a deployment to 'github-pages' at commit '1f8d31a', also by 'coursar' 11 minutes ago, with an 'Active' status. Another 'View deployment' link is present to the right of this entry.

Deployed to github-pages

[1f8d31a](#) was deployed by **coursar** 11 minutes ago Active [View deployment](#)

Activity log Show: All environments ▾

 **github-pages** at 1f8d31a  
Deployed by **coursar** 11 minutes ago Active [View deployment](#)

Обратите внимание: эта вкладка доступна только владельцу репозитория и collaborator'ам.

Поэтому **не забывайте вставлять URL развёрнутого сайта в README.md.**



# ВЫБОР КОНФИГУРАЦИИ

Вы можете выбрать любой образ из числа доступных на Appveyor:

- [Windows-образы](#)
- [Linux-образы](#)



# ПОЛНОЕ ОПИСАНИЕ .APPVEYOR.YML

<https://www.appveyor.com/docs/appveyor-yml/>

# APPVEYOR.YML ИЛИ .APPVEYOR.YML?

Допустимо любое из этих названий. Мы в рамках курса будем использовать `.appveyor.yml`.

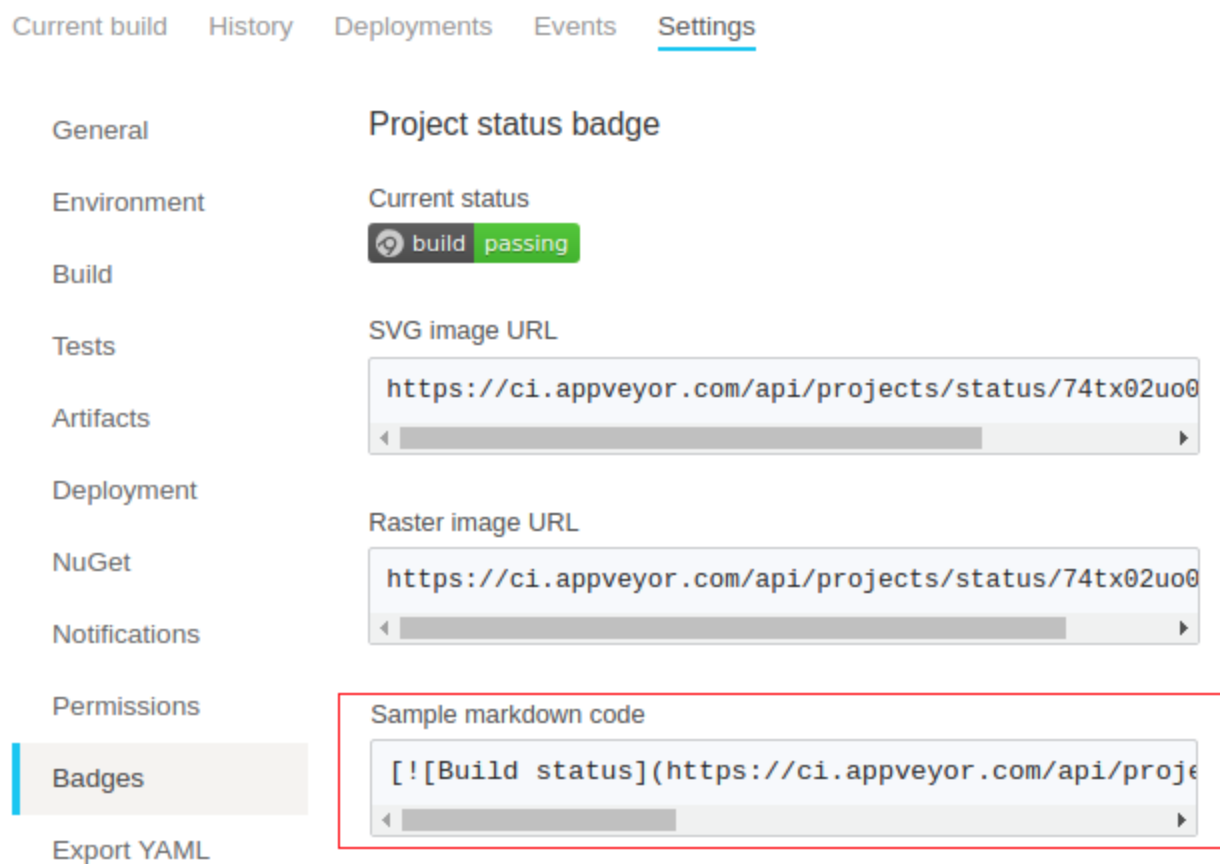


## ПРИМЕРЫ .APPVEYOR.YML

- [Webpack](#)
- [React](#)
- [Jest](#)

# STATUS BADGE

На странице **Settings** → **Badges** Appveyor предлагает код для “бейджика” статуса вашего проекта:



Current build History Deployments Events Settings

General

Environment

Build

Tests

Artifacts

Deployment

NuGet

Notifications


Permissions

**Badges**

Export YAML

### Project status badge

Current status

 build passing

SVG image URL

`https://ci.appveyor.com/api/projects/status/74tx02uo0`

Raster image URL

`https://ci.appveyor.com/api/projects/status/74tx02uo0`

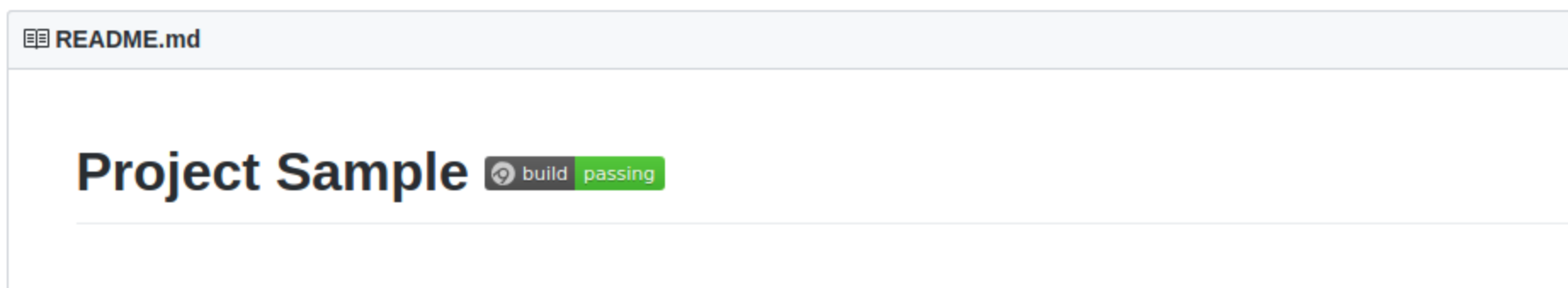
Sample markdown code

```
[![Build status](https://ci.appveyor.com/api/projects/status/74tx02uo0)]
```



# STATUS BADGE

Этот badge необходимо разместить в файле README.md для отображения текущего статуса вашего проекта:



Обратите внимание, что есть некая задержка при обновлении бейджика

# ИТОГИ

Сегодня мы с вами рассмотрели достаточно много важных вещей, а именно:

- yarn — возможность использования альтернативного менеджера пакетов
- Continuous Deployment — автоматизация развёртывания приложения

Начиная с сегодняшнего дня мы будем требовать, чтобы все ваши домашние задания были подключены к AppVeyor (не забывайте про бейджики), если только в описании задания явно не указано иное.



**Задавайте вопросы и напишите отзыв о лекции!**

**АЛЕКСАНДР ШЛЕЙКО**

 [a.shleyko@yandex.ru](mailto:a.shleyko@yandex.ru)

 [vk.com/shleiko](https://vk.com/shleiko)

 [@dustyo\\_0](https://t.me/@dustyo_0)