**Совместная разработка в команде на GitHub**

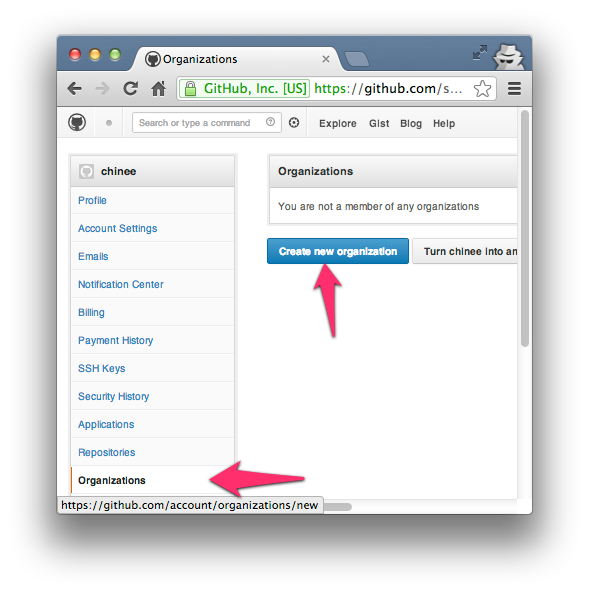
**Инструмент 1: Добавление членов команды**

Как правило, существует два способа настройки Github для совместной работы:

1. **Организации**. Владелец организации может создавать множество команд с разными уровнями доступа для различных репозиториев
2. **Сотрудники**. Владелец репозитория может добавлять коллабораторов с доступом Read + Write для одного репозитория

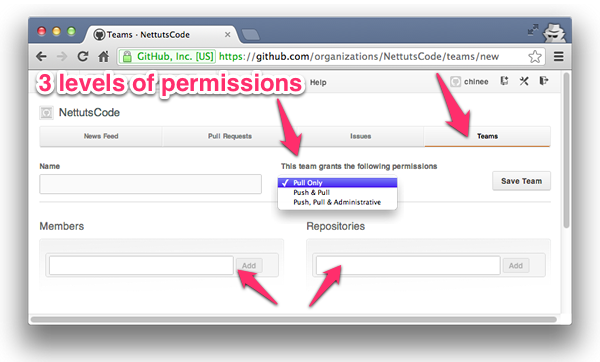
**Organizations**

Если вы контролируете несколько команд и хотите установить разные уровни доступа для каждой команды с различными членами и добавить каждого участника в разные репозитории, то организация будет для вас наилучшим вариантом. Любая учетная запись пользователя Github уже может создавать бесплатные организации для репозиториев с открытым исходным кодом. Чтобы создать организацию, просто перейдите [**на страницу настроек своей организации**](https://github.com/settings/organizations):



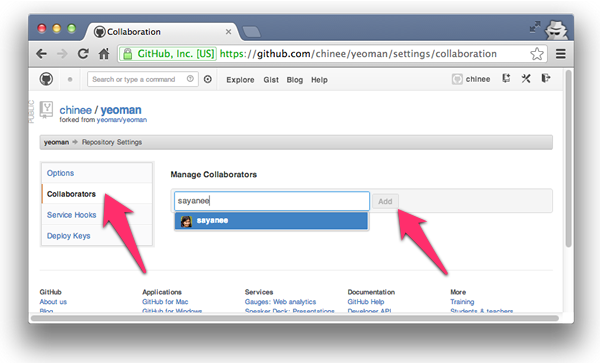
Чтобы получить доступ к странице команд для вашей Организации, вы можете просто перейти на http://github.com/organizations/[organization-name]/teams, чтобы просмотреть их или даже посетить https://github.com/organizations/[organization-name]/teams/new Для создания новых команд с тремя уровнями доступа, такими как:

1. **Pull Only**: [**выборка и слияние**](https://www.kernel.org/pub/software/scm/git/docs/git-pull.html) с другим репозиторием или локальной копией. Доступ только для чтения.
2. **Push and Pul**l: (1) наряду с [**обновлением**](https://www.kernel.org/pub/software/scm/git/docs/git-push.html) удаленного репозитория. Читайте + Запись.
3. **Push, Pull & Administrative**: (1), (2) наряду с правами на информацию о выставлении счетов, созданием команд, а также удаление аккаунтов организации. Чтение + запись + доступ администратора



**Соавторы**

Коллабораторы (соавторы) используются для предоставления возможности **«читать + писать»** в один репозиторий, принадлежащий личной учетной записи. Чтобы добавить [**Collaborators**](https://help.github.com/articles/how-do-i-add-a-collaborator) (другие личные учетные записи Github), перейдите на страницу https://github.com/[username]/[repo-name]/settings/collaboration:



После этого каждый соавтор увидит изменение статуса доступа на странице репозитория. После того, как у нас есть доступ на запись к репозиторию, мы можем сделать git clone, поработать над изменениями, сделать git pull для извлечения и слияния любых изменений в удаленном репозитории и, в конечном счете, git push, для обновления удаленного репозитория с собственными изменениями:



**Инструмент 2: Pull Requests**

[**Pull Requests**](https://help.github.com/articles/using-pull-requests) - отличный способ внести свой вклад в репозиторий, сделав его форк. В конце дня, если мы хотим, мы можем отправить pull request владельцу репозитория, чтобы объединить наши изменения кода. Сам pull request может включать обсуждение качества кода, функций или даже общей стратегии.

Давайте теперь рассмотрим основные шаги для **[pull request](https://help.github.com/articles/using-pull-requests)**.

**Инициирование pull request**

В Github есть две модели для **[pull request](https://help.github.com/articles/using-pull-requests)**:

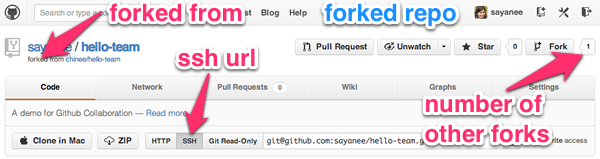
1. **Модель Fork & Pull** - используется в общедоступном репозитории, на который у вас нет push-доступа
2. **Share Repository Model** - используется в частном репозитории, на который у нас есть push-доступ. В этом случае форк не требуется.

Здесь мы видим рабочий процесс между двумя пользователями (repo-owner и forked-repo-owner) для модели Fork and Pull:

1. Определите репозиторий Github, в который вы хотите внести свой вклад, и нажмите кнопку «Fork», чтобы создать клон репозитория в вашей собственной учетной записи Github:



1. Это создаст точную копию репозитория в вашем собственном аккаунте



1. [**Выберите URL-адрес SSH**](https://help.github.com/articles/why-is-git-always-asking-for-my-password), чтобы он запрашивал вашу парольную кодовую фразу SSH вместо имени пользователя и пароля каждый раз, когда вы делаете git push или git pull. Затем мы будем клонировать этот репозиторий на наш локальный компьютер:

|  |  |
| --- | --- |
| 1  2 | $ git clone [ssh-url] [folder-name]  $ cd [folder-name] |

1. Как правило, мы создадим новую ветку git для каждой новой задачи. Это хорошая практика, потому что в будущем, если мы продолжим обновление ветки после некоторых обсуждений, [**запрос на перенос будет автоматически обновляться**](https://stackoverflow.com/questions/9790448/how-to-update-a-pull-request). Давайте создадим новую ветку, чтобы внести очень простое изменение в файл readme.md:

|  |  |
| --- | --- |
| 1 | $ git checkout -b [new-feature] |

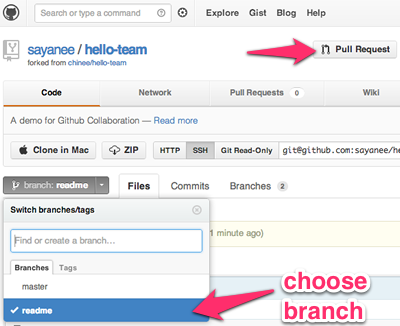
1. После внесения соответствующих дополнений для создания новых функций мы просто передадим новые изменения и проверку в ветку git master:

|  |  |
| --- | --- |
| 1  2  3 | $ git add .  $ git commit -m "information added in readme"  $ git checkout master |

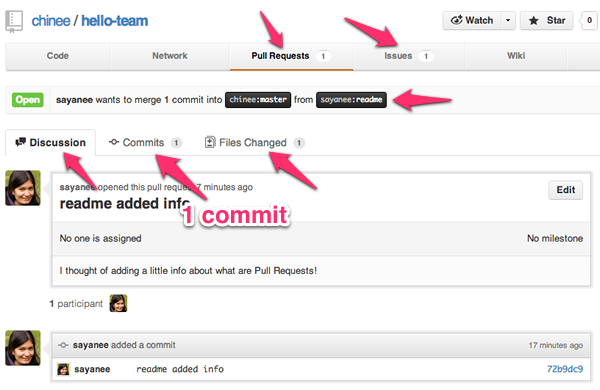
1. На этом этапе мы запушим ветку в удаленный репозиторий. Для этого мы сначала перейдем на ветку с новой задачей, а так же на псевдоним для удаленного репозитория. Затем мы будем пушить изменения с помощью git push [git-remote-alias] [branch-name]:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | $ git branch  \* master  readme  $ git remote -v  origin  git@github.com:[forked-repo-owner-username]/[repo-name].git (fetch)  origin  git@github.com:[forked-repo-owner-username]/[repo-name].git (push)  $ git push origin readme |

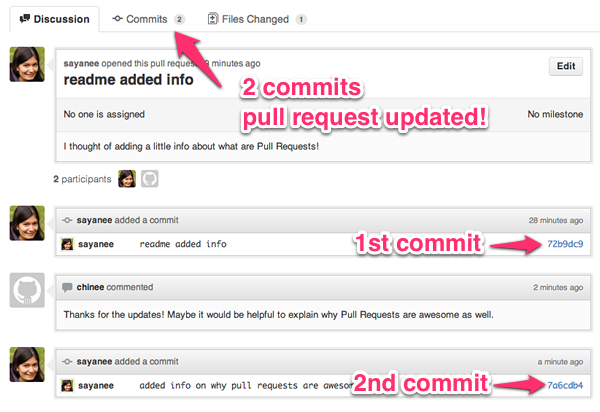
1. На нашей развязанной странице Github репозитория мы перейдем к ветке с новой функцией, а затем нажмите кнопку «Pull Request».



1. После отправки пул реквеста он напрямую приведет нас к странице запроса в исходном репозитории. Мы увидим наш запрос на pull.



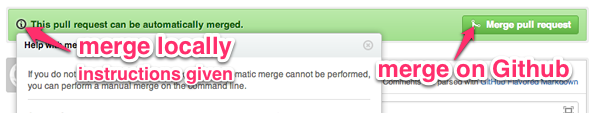
1. После обсуждения возможно, что владелец форкнутого репозитория может захотеть добавить изменения в новую функцию. В этом случае мы выберем одну и ту же ветку на нашей локальной машине, зафиксируем ее и запушим ее обратно на Github. Когда мы заходим на страницу запроса в оригинальном репозитории, он будет автоматически обновляться!



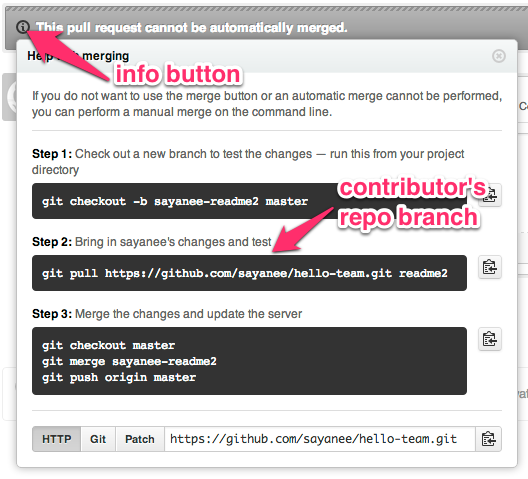
**Слияние пул реквеста**

Если вы являетесь владельцем оригинального репозитория, существует [**два способа слияния**](https://help.github.com/articles/merging-a-pull-request) входящего пул реквеста:

1. **Слияние непосредственно на Github**: если мы делаем слияние непосредственно на Github, то убедитесь, что нет конфликтов, и реквест готов к объединению в ведущую ветку. Владелец оригинального хранилища может просто щелкнуть зеленую кнопку «Слить запрос», чтобы сделать это:



1. **Слияние на наших локальных машинах**: в других случаях могут возникнуть конфликты слияния, и после нажатия кнопки «Информация» у Github будут четкие инструкции о том, как мы можем объединить ветвь на нашей локальной машине, потянув за изменения из ветви контрибьютера:



Существуют различные модели создания веток, используемые для управления версиями в командах разработки программного обеспечения. Вот две популярные модели рабочего процесса git: (1) [**рабочий процесс Github**](http://scottchacon.com/2011/08/31/github-flow.html), который имеет простую ветвящуюся модель и использует запросы на pull, и (2) **[Gitflow](http://nvie.com/posts/a-successful-git-branching-model/)**, который имеет более обширное разветвление. Модель, которая в конечном итоге будет выбрана, определенно будет меняться в зависимости от команды, проекта и ситуации.

**Инструмент 3: Отслеживание ошибок**

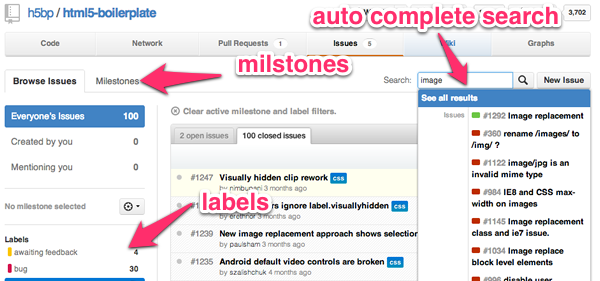
[***Pull Requests***](https://help.github.com/articles/using-pull-requests)*- отличный способ внести свой вклад в репозиторий сделав его форк.*

В Github центр для отслеживания ошибок - это issues. Несмотря на то, что они в основном предназначены для отслеживания ошибок, также полезно использовать «Issues» следующими способами:

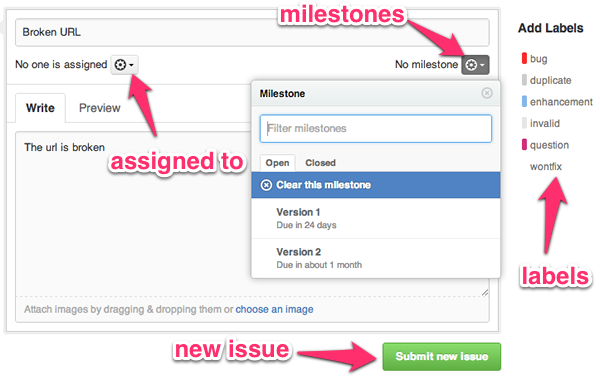
* **Ошибки**: вещи, которые явно сломаны и нуждаются в исправлениях
* **Особенности**: Удивительные крутые новые идеи для реализации
* **Список дел**: контрольный список задач для завершения

Давайте рассмотрим некоторые особенности проблем:

1. **Labels**: цветные категории для каждого issue. Они полезны для фильтрации.
2. **Milestones**: они относятся к категориям, которые могут быть связаны с каждым issue, и полезны для определения того, какие проблемы необходимо обрабатывать для следующего релиза. Кроме того, поскольку этапы связаны с проблемами, то автоматически обновляется индикатор выполнения при закрытии каждой связанной проблемы.
3. **Search**: Автокомплит поиска для issues и milestones



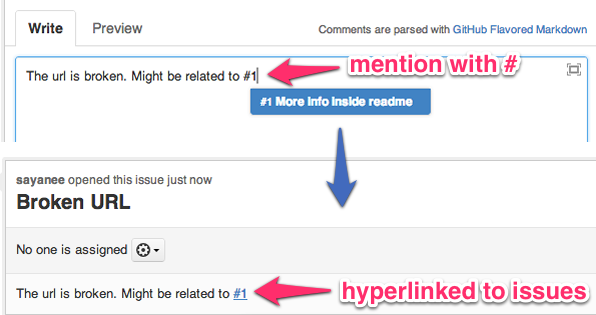
1. **Assignment**: каждый вопрос может быть назначен ответственному лицу для исправления проблемы. Еще одна полезная функция - посмотреть, над чем мы должны работать.



1. **Автоматическое закрытие**: сообщения в коммитах вида Fixes/Fixed or Close/Closes/Closed #[issue-number] автоматически закроют issue.

|  |  |
| --- | --- |
| 1  2  3 | $ git add .  $ git commit -m "corrected url. fixes #2"  $ git push origin master |

1. 
2. **Mentions**: каждый может оставить примечание, просто указывая #[номер issue] в своих сообщениях. Поскольку номера issue являются гиперссылками, это позволяет легко упоминать связанные issue во время обсуждения.



**Инструмент 4: Аналитика**

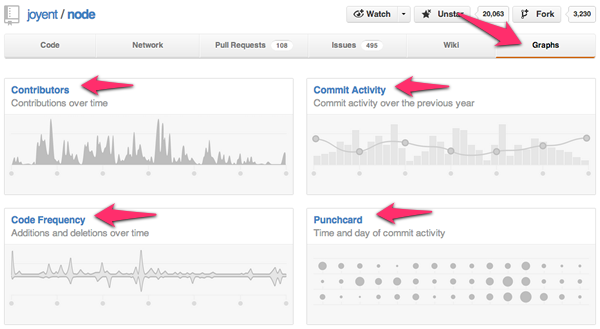
*Понятно, что мы можем тесно связать наш список задач и обновления с нашими кодами.*

Есть два инструмента, которые дают представление о репозитории - Graphs and Network. [**Графики Github**](https://github.com/blog/1093-introducing-the-new-github-graphs) отображает соавторов и их коммиты, в то время как **[Github Network](https://github.com/blog/39-say-hello-to-the-network-graph-visualizer)** обеспечивает визуализацию для каждого участника. Эти аналитики и графики становятся очень мощными, особенно при работе в командах.

**Графики**

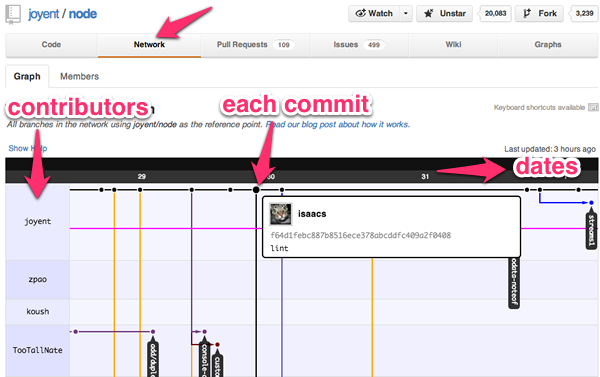
Графики предоставляют подробную аналитику, такую как:

* **Contributors**: кто был соавтором? И сколько строк кода они добавляли или удаляли?
* **Commit Activity**: В какие недели произошли фиксации в прошлом году?
* **Code Frequency:** сколько строк кода было зафиксировано на протяжении всего жизненного цикла проекта?
* **Punchcard:** В какое время дня обычно совершаются коммиты?



**Network**

[**Github Network**](https://github.com/blog/39-say-hello-to-the-network-graph-visualizer) - это мощный инструмент, который позволяет вам видеть, как коммитит каждый участник и как они связаны друг с другом. Когда мы смотрим на визуализатор целиком, мы видим каждую фиксацию в каждой ветви каждого репозитория, принадлежащего сети. Очень круто!



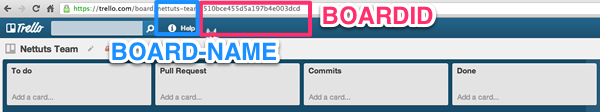
**Инструмент 5: Управление проектами**

В то время как issues Github имеют возможности управления проектами с помощью Issues и Milestones, некоторые команды могут предпочесть другой инструмент из-за других функций или существующего рабочего процесса. В этом разделе мы увидим, как мы можем связать Github с двумя другими популярными инструментами управления проектами - **[Trello](https://trello.com/)** и **[Pivotal Tracker](https://www.pivotaltracker.com/)**. С помощью сервисов Github мы можем автоматизировать задачу обновления с помощью коммитов, проблем и многих других действий. Эта автоматизация помогает не только экономить время, но и повышает точность обновлений для любой команды разработчиков программного обеспечения.

**Github и Trello**

Trello обеспечивает простой, визуальный способ управления задачами. Используя **[Agile Software Development](https://en.wikipedia.org/wiki/Agile_software_development)**, карточки Trello могут эмулировать простую виртуальную **[Kanban Board](https://en.wikipedia.org/wiki/Kanban_board)**. В качестве примера мы автоматически создадим карточку Trello всякий раз, когда будет появляться новый пул реквест с помощью Github Service Hooks. Давайте пройдем через все шаги!

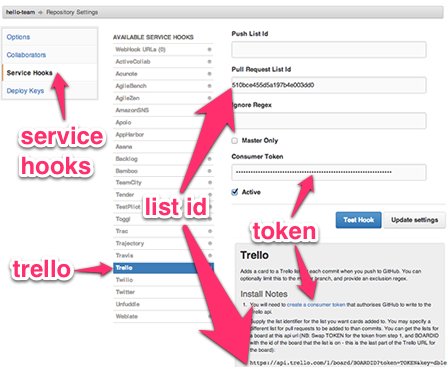
1. Откройте свой аккаунт в Trello, если у вас его еще нет, и создайте новую доску Trello.



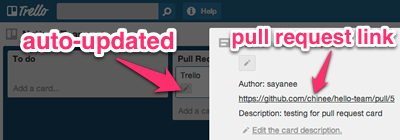
1. Перейдите в репозиторий Github> Настройки> Хуки для сервисов> Trello
2. Получите TOKEN в разделе «Примечания по установке» № 1 с гиперссылкой, предоставленной для аутентификации.
3. В разделе Примечания по установке #2 используйте URL-адрес для создания форматированной в json структуры, которая дает нам list id для каждой карточки Trello. BOARDID является частью URL-адреса, когда мы посещаем доску на странице https://trello.com/board/[BOARD-NAME]/[BOARDID]. TOKEN можно создать с помощью гиперссылки, указанной в разделе Примечания по установке #1.



1. Вернитесь в сервисные хуки Github, введите list id и token. Установите Active, Test Hook, и теперь все настроено на автоматическое обновление каждый раз, когда будет новый пул реквест.



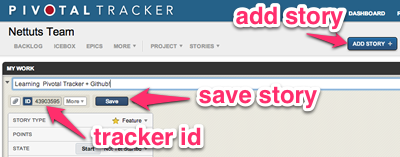
1. В следующий раз, когда появится запрос, у карточки Trello Pull Request автоматически будет новый элемент!



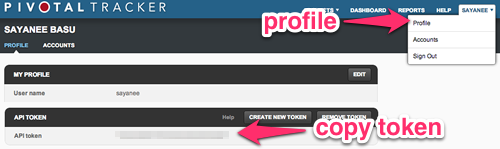
**Github и Pivotal Tracker**

[**Pivotal Tracker**](https://www.pivotaltracker.com/) - еще один легкий гибкий инструмент управления проектами, в котором планирование на основе истории позволяет команде легко взаимодействовать, мгновенно реагируя на различные изменения и прогресс проекта. Основываясь на текущем прогрессе команды, он также может создавать диаграммы для анализа скорости команды, итераций разработки, релизов и прочего. В этом кратком примере мы автоматически доставим историю, связав ее с коммитом на Github!

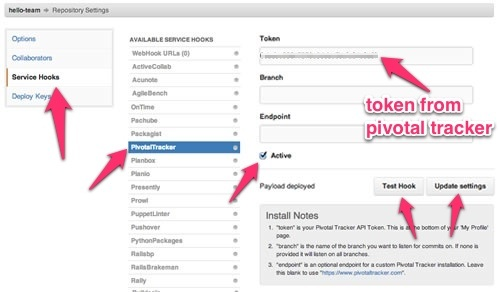
1. Создайте новый проект в Pivotal Tracker с новой Story  которая должна быть сделана.



1. Перейдите в профиль> API-токен (справа внизу). Скопируйте указанный API токен.



1. Вернитесь в репозиторий Github> Настройки> Хуки для сервисов> Pivotal Tracker. Вставьте токен, установите флажок Активно и нажмите «Обновить настройки». Теперь все настроено на автоматическое предоставление данных для Pivotal Tracker Stories из коммитов на Github!



1. Наконец, мы зафиксируем наши изменения и [**добавим идентификатор трекера**](http://pivotallabs.com/level-up-your-development-workflow-with-github-pivotal-tracker/) в сообщение фиксации в формате git commit -m "message [delivers #tracker\_id]"

|  |  |
| --- | --- |
| 1  2  3 | $ git add .  $ git commit -m "Github and Pivotal Tracker hooks implemented [delivers #43903595]"  $ git push |

1. Теперь, когда мы вернемся к Pivotal Tracker, мы увидим, что история была автоматически доставлена со ссылками на конкретный коммит Github, который показывает изменения файла!



С примерами Trello и Pivotal Tracker ясно, что мы можем тесно связать наш список задач и обновления с нашими коммитами. Это огромная экономия времени при работе в команде, и это повышает точность при связывании задач с точными фиксациями. Хорошей новостью является то, что если вы уже используете другие инструменты управления проектами, такие как **[Asana](https://asana.com/)**, **[Basecamp](https://basecamp.com/)** и другие, вы также можете создать Service Hooks аналогичным образом. Если для вашего текущего инструмента управления проектами нет существующих сервисных хуков, вы даже [**можете их создать сами**](https://github.com/github/github-services)!

**Инструмент 6: Непрерывная интеграция**

[**Непрерывная интеграция**](https://en.wikipedia.org/wiki/Continuous_integration) (CI) является важной частью всех проектов разработки программного обеспечения, с которыми работают команды разработчиков. CI гарантирует, что, когда разработчик выкатывает свой код, автоматическая сборка (включая тесты) быстро обнаруживает ошибки интеграции. Это определенно уменьшает ошибки интеграции и делает быструю итерацию намного более эффективной. В этом примере мы увидим, как можно использовать **[Travis CI](https://travis-ci.org/)** вместе с Github для CI для обнаружения ошибок, а также для рекомендаций слияния, когда проходят все тесты.

**Настройка Travis CI**

Мы будем использовать простой проект «hello world» для [**node.js**](https://nodejs.org/) вместе с [**grunt.js**](https://gruntjs.com/) в качестве инструмента сборки для настройки Travis CI проекта. Вот файлы, находящиеся в проекте:

1. Файл hello.js - это проект nodejs. Здесь мы намеренно не будем оставлять точку с запятой, чтобы грант не смог собрать билд:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | var http = require('http');  http.createServer(function (req, res) {  res.writeHead(200, {'Content-Type': 'text/plain'});    res.end('Hello World in Node!\n') // without semicolon, this will not pass linting  }).listen(1337, '127.0.0.1');  console.log('Server running at [**http://127.0.0.1:1337/**](http://127.0.0.1:1337/)'); |

1. package.json определяет зависимости:

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12 | {    "name": "hello-team",    "description": "A demo for github and travis ci for team collaboration",    "author": "name <email@email.com>",    "version": "0.0.1",    "devDependencies": {      "grunt": "~0.3.17"    },    "scripts": {      "test": "grunt travis --verbose"    }  } |

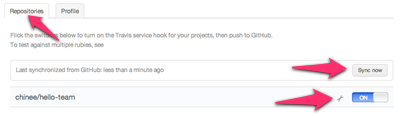
1. Инструмент сборки gruntjs имеет для простоты только одну задачу (линтинг) :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | module.exports = function(grunt) {      grunt.initConfig({       lint: {        files: ['hello.js']      }    });    grunt.registerTask('default', 'lint');    grunt.registerTask('travis', 'lint');  }; |

1. .travis.yml - это файл конфигурации Travis, который заставит Travis запускать наши тесты:

|  |  |
| --- | --- |
| 1  2  3 | language: node\_js  node\_js:    - 0.8 |

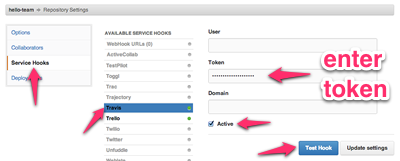
1. Затем войдите в Travis со своей учетной записью Github и включите привязку репозитория под вкладкой репозитория.



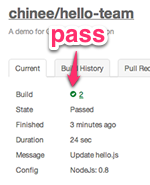
1. Если прошлый шагне вызывает сборку, нам придется вручную настроить сервисный хук. Чтобы настроить его вручную, скопируйте токен под вкладкой профиля.



1. Вернитесь в репозиторий Github, чтобы настроить сервисные хуки Github Travis с помощью токена.



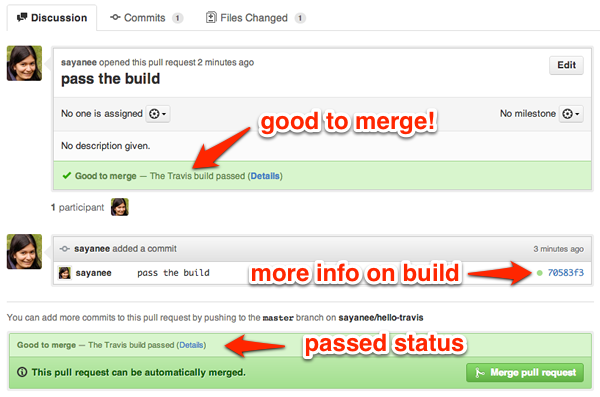
1. Сначала нам нужно сделать вручную git push для запуска первой сборки Travis, и если все будет в порядке, просто посетите http://travis-ci.org/[username]/[repo-name], чтобы увидеть что сборка прошла!



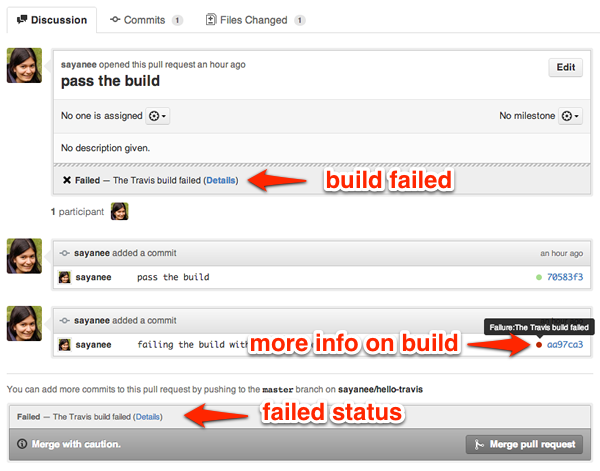
**Travis CI и пул реквесты**

Раньше, без какого-либо CI в процессе пул реквеста, этапы выполнялись примерно так: 1) создание запроса (2) слияние (3), тестируем все ли работает. Когда Travis CI подключится к пул реквесам, мы сможем инвертировать шаги 2 и 3, что еще больше ускорит принятие решений о том, следует ли сливать изменения, так как Travis даст нам статус билда для каждого пул реквеста. Давайте посмотрим, как это сделать.

1. Отправьте пул реквест с успешным билдом, и Travis сделает свою магию, чтобы вы знали, что слияние будет успешным еще даже до самого слияния



1. Если билд с этим пул реквестом падает, Travis также предупредит вас.

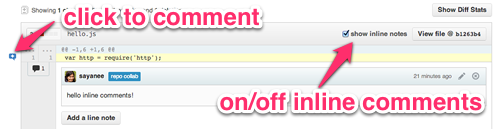


1. Если мы нажмем на кнопку красного предупреждения, она также будет ссылаться на страницу travis, чтобы показать нам детали сборки.

Travis CI с Github чрезвычайно полезен для команд из-за автоматических сборок и немедленного уведомления. Это, безусловно, делает цикл исправления ошибок намного короче. Если вы используете **[Jenkins](https://jenkins-ci.org/)**, еще один популярный инструмент CI, то вы тоже можете настроить сервисные хуки Github.

**Инструмент 7: Обзор кода**

С каждой фиксацией изменений Github позволяет использовать чистый интерфейс для общих комментариев или даже конкретных комментариев к отдельной строчке кода. Возможность делать комментарии или задавать вопросы по каждой отдельной строке кода очень полезна при проведении обзоров строка за строкой. Чтобы просмотреть встроенные комментарии, установите флажок в верхней части каждой фиксации.



Давайте рассмотрим некоторые шаблоны URL-адресов, которые могут быть использованы, чтобы помочь нам в обзоре кода, быстро предоставив нам различия между фиксациями:

1. **Compare branches / tags / SHA1**: используйте шаблон URL https://github.com/[username]/[repo-name]/compare/[starting-SHA1]...[ending-SHA1]. Вы можете сделать то же самое с веткой или тегами.



1. **Сравнение без пробелов**: добавьте ?w=1 для сравнения URL-адресов



1. **Diff**: добавьте .diff к URL-адресам сравнения, чтобы получить информацию о git diff в текстовом формате. Это может быть полезно для сценариев.
2. **Patch**: добавьте .patch к URL-адресам сравнения, чтобы получить информацию о git diff, отформатированную для[**отправки исправлений электронной почты**](https://www.kernel.org/pub/software/scm/git/docs/git-format-patch.html).
3. **Связывание строк**. Когда мы нажимаем номер строки в любом файле, Github добавит #line в конце URL-адреса и сделает весь цвет фона строки желтым. Это очень удобно для указания конкретной строки. GitHub также принимает диапазоны строк, добавляя #start-end. Ниже приведены [**примеры связывания**](https://github.com/NETTUTS/team-collaboration-github/blob/master/.travis.yml#L4) строк и [**диапазонов строк**](https://github.com/NETTUTS/team-collaboration-github/blob/master/.travis.yml#L2-3).

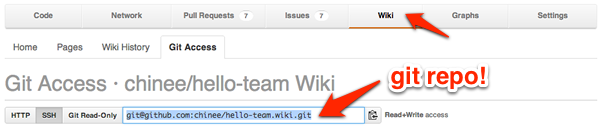
**Инструмент 8: документация**

В этом разделе мы рассмотрим два метода создания документации:

1. **Формальная документация**: Github Wiki для создания документации для исходного кода
2. **Неофициальная документация**: **[Github Hubot](https://github.com/github/hubot)** документирует обсуждения между членами команды и автоматизирует поиск информации, взаимодействуя с бот-чатом!
3. **Упоминания, ярлыки и эмози**

**Github Wiki**

В каждом репозитории может быть создана Github Wiki, и это может оказаться очень удобно иметь в одном репозитории как исходный код, так и документацию для него. Чтобы создать Wiki, просто перейдите на вкладку Wiki в главном заголовке, и вы готовы к созданию страниц с информацией. Сама Wiki имеет собственное управление версиями, и данные могут быть клонированы на наш локальный компьютер для обновлений или даже автономного доступа.



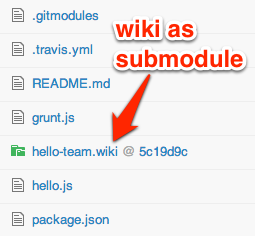
Одна вещь, которую я нахожу очень полезной, - это интеграция Github Wiki в основной проект исходного кода, так что мне не нужно поддерживать два отдельных проекта git. Для этого я добавляю Wiki git repo как **[субмодуль](https://git-scm.com/book/ch6-6.html)** из ветки master. Если вы используете Travis CI или любой другой CI, убедитесь, что инструмент [**сборки игнорирует подмодуль wiki**](https://github.com/travis-ci/travis-build/pull/46). Для файла Travis CI .travis.yml добавьте следующее:

|  |  |
| --- | --- |
| 1  2 | git:    submodules: false |

Затем добавьте wiki-подмодуль git в основной репозиторий кода:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | $ git submodule add git@github.com:[username]/[repo-name].wiki.git  Cloning into 'hello-team.wiki'...  remote: Counting objects: 6, done.  remote: Compressing objects: 100% (3/3), done.  remote: Total 6 (delta 0), reused 0 (delta 0)  Receiving objects: 100% (6/6), done.  $ git add .  $ git commit -m "added wiki as submodule"  $ git push origin master |

Теперь Wiki будет выглядеть как подмодуль в основном проекте с исходным кодом.



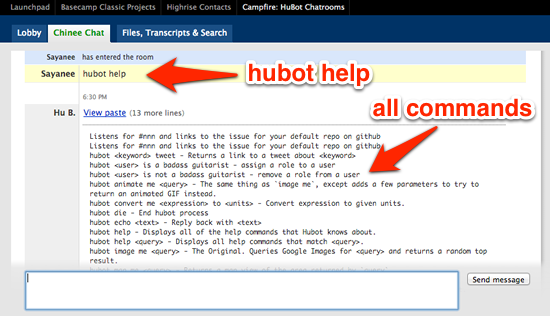
**Github Hubot**

*Если коротко, Hubot может сделать процесс ведения документации гораздо приятнее, добавляя уведомление командных обсуждений о важных коммитах.*

[**Hubot**](https://github.com/github/hubot) - это просто бот-чат, который может получать информацию или предоставлять уведомление при подключении к Github коммитам, issues или активности. В команде, которая стремится значительно сократить количество встреч или даже полностью устранить их, Hubot с интерфейсом чата среди членов команды помогает документировать каждую отдельную дискуссию. Это, безусловно, способствует гибкому графику работы, так как никто не должен присутствовать одновременно для обсуждения. Предупреждение: Hubot ужасно вызывает привыкание!

Итак давайте начнем с настройки Hubot, размещенным на **[Heroku](https://www.heroku.com/)**, и ботом с интерфейсом чата **[Campfire](http://campfirenow.com/)**! Для обоих и Heroku и Campfire есть бесплатные версии, с которым можно начать.

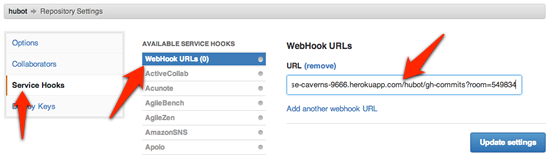
1. Мы воспользуемся [**версией Hubot от Github's Campfire**](https://github.com/github/hubot). Если вы хотите, вы можете выбрать [**адаптеры для других чатов**](https://github.com/github/hubot/wiki), таких как Skype, IRC, Gtalk и т.д.
2. Откройте новую учетную запись Campfire только для Hubot, и эта учетная запись создаст новую комнату для чата, в которую все остальные будут приглашены позже.
3. Разверните Hubot на Heroku с [**помощью этой инструкции**](https://github.com/github/hubot/wiki/Deploying-Hubot-onto-Heroku), приведенной на Hubot Wiki. Не волнуйтесь, если URL-адрес приложения heroku дает сообщение Can not GET /, поскольку по [**умолчанию еще нечего делать**](https://github.com/github/hubot/issues/286).
4. С учетной записью Hubot Campfire пригласите себя. Теперь войдите в свою учетную запись Campfire и выполните команду Hubot help. Эта команда предоставит вам все доступные команды для hubot.



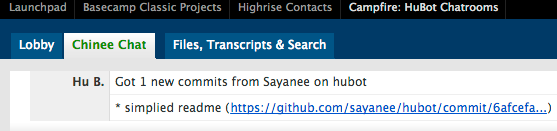
1. Попробуйте что-нибудь, например, hubot ship it или hubot map me CERN.



1. Затем добавим скрипт Hubot. Существует [**множество доступных**](https://github.com/github/hubot-scripts/tree/master/src/scripts) [**команд с иллюстрациями**](https://hubot-script-catalog.herokuapp.com/).
2. В качестве примера мы добавим скрипт **[github-commits](https://github.com/github/hubot-scripts/blob/master/src/scripts/github-commits.coffee)**, чтобы каждый раз, когда есть фиксация, Hubot уведомит нас об этом в чате. Поместите файл github-commits.coffee внутри папки scripts.
3. Обновите файл package.json с соответствующими зависимостями, как указано в начале каждого файла сценария hubot, в комментариях.
4. Разверните изменения на Heroku еще раз с помощью git push heroku master.
5. Перейдите в репозиторий на Github, чье уведомление об фиксации будет отображаться в чате. Добавьте веб-хук в настройки репозитория. В случае с указанным сценарием «github-commit» webhook будет [HUBOT\_URL]:[PORT]/hubot/gh-commits?Room=[ROOM\_ID]



1. В следующий раз, когда в репозитории будет фиксация, Hubot будет об этом говорить и писать в чат!



Ознакомьтесь [**с другими скриптами Hubot, связанными с Github**](https://github.com/github/hubot-scripts), или если вы хотите написать один, есть крутой [**учебник по этому поводу**](https://net.tutsplus.com/tutorials/javascript-ajax/writing-hubot-plugins-with-coffeescript/)! Короче говоря, Hubot может значительно добавить веселья в документировании и уведомлении командных дискуссий о важных коммитах, проблемах и действиях, происходящих с нашими репозиториями на Github. Попробуйте его!

В качестве заключительной заметки о командной работе на Github, вот несколько советов по производительности:

1. **Упоминания**. В любой текстовой области мы можем указать другого пользователя github с @user, и пользователь получит уведомление о комментарии
2. **Клавиши быстрого доступа** - нажмите [shift] +? Для доступа к клавишам быстрого доступа Github на любой странице.
3. **Emoji** - Используйте [**список Emoji**](http://www.emoji-cheat-sheet.com/), Github textareas также поддерживает вставку этих значков. Попробуйте их и повеселитесь со своими товарищами по команде!

**Использование Github не для разработки**

Большинство думает использовать Github только для программных проектов. В конце концов, Github был создан специально для создания кода. Но есть некоторые классные репозитории Github, которые используются для проектов, не связанных с кодом, и они были одинаково великолепны для сотрудничества и обсуждения. Поскольку эти проекты также доступны с открытым исходным кодом, и каждый может внести свой вклад, быстро исправлять ошибки, легко сообщать об ошибках и эффективно сотрудничать с единомышленниками. Просто ради интереса, вот некоторые из них:

* **Исправления для дома:**отслеживание проблем в качестве инструмента мониторинга
* **Книги:** [**Маленькая книга по MongoDB**](https://github.com/karlseguin/the-little-mongodb-book), [**Основы Backbone**](https://github.com/addyosmani/backbone-fundamentals)
* **Тексты песен:**JSConfEU Тексты песен
* **Найти друга:** **[boyfriend\_require](https://github.com/norinori2222/boyfriend_require/blob/master/README-en.md)**
* **Наставничество:** **[Wiki](https://github.com/dianakimball/mentoring)**
* **Геномические данные:**эпидемия Ash Dieback
* **Блоги:** [**CSS Wizardry**](https://github.com/csswizardry/csswizardry.github.com)

И интересно, что думает об этом команда **[Github](https://news.ycombinator.com/item?id=4963433)**?

*«Мы кайфуем от использования GitHub»*

**Дополнительные ресурсы**

* [**Социальная разработка на GitHub**](https://www.cs.cmu.edu/~xia/resources/Documents/cscw2012_Github-paper-FinalVersion-1.pdf), исследовательская работа Университета Карнеги-Меллона
* [**Как Github использует Github для создания Github**](http://zachholman.com/talk/how-github-uses-github-to-build-github/) Зак Холман
* [**Секреты Git и Github**](http://zachholman.com/talk/git-github-secrets/) Зак Холман
* [**Новые возможности в Github**](https://github.com/blog/category/ship)из блога Github
* Github Help: [**пул реквесты**](https://help.github.com/articles/using-pull-requests), **[форки репозитория](https://help.github.com/articles/fork-a-repo)**
* [**Возможности Github для совместной работы**](https://github.com/features/projects)
* Учебники Nettuts +: **[Git](https://net.tutsplus.com/tag/git/)** и **[Github](https://net.tutsplus.com/tag/github/)**
* [**Повелитель файлов: как Github приручил бесплатное программное обеспечение (и многое другое)**](https://www.wired.com/wiredenterprise/2012/02/github/) Wired