

- [About](#)
 - [Branching and Merging](#)
 - [Small and Fast](#)
 - [Distributed](#)
 - [Data Assurance](#)
 - [Staging Area](#)
 - [Free and Open Source](#)
 - [Trademark](#)
- [Documentation](#)
 - [Reference](#)
 - [Book](#)
 - [Videos](#)
 - [External Links](#)
- [Downloads](#)
 - [GUI Clients](#)
 - [Logos](#)
- [Community](#)

This book is available in [English](#).

Full translation available in

[azərbaycan dili](#),
[български език](#),
[Deutsch](#),
[Español](#),
[Français](#),
[Ελληνικά](#),
[日本語](#),
[한국어](#),
[Nederlands](#),
[Русский](#),
[Slovenščina](#),
[Tagalog](#),
[Українська](#)
[简体中文](#),

Partial translations available in

[Čeština](#),
[Македонски](#),
[Polski](#),
[Српски](#),
[Ўзбекча](#),
[繁體中文](#),

Translations started for

[Беларуская](#),
[فارسی](#),

[Indonesian](#),
[Italiano](#),
[Bahasa Melayu](#),
[Português \(Brasil\)](#),
[Português \(Portugal\)](#),
[Svenska](#),
[Türkçe](#).

The source of this book is [hosted on GitHub](#).
Patches, suggestions and comments are welcome.

[Chapters ▾](#)

1. [1. Введение](#)

1. 1.1 [О системе контроля версий](#)
2. 1.2 [Краткая история Git](#)
3. 1.3 [Что такое Git?](#)
4. 1.4 [Командная строка](#)
5. 1.5 [Установка Git](#)
6. 1.6 [Первоначальная настройка Git](#)
7. 1.7 [Как получить помощь?](#)
8. 1.8 [Заключение](#)

2. [2. Основы Git](#)

1. 2.1 [Создание Git-репозитория](#)
2. 2.2 [Запись изменений в репозиторий](#)
3. 2.3 [Просмотр истории коммитов](#)
4. 2.4 [Операции отмены](#)
5. 2.5 [Работа с удалёнными репозиториями](#)
6. 2.6 [Работа с тегами](#)
7. 2.7 [Псевдонимы в Git](#)
8. 2.8 [Заключение](#)

3. [3. Ветвление в Git](#)

1. 3.1 [О ветвлении в двух словах](#)
2. 3.2 [Основы ветвления и слияния](#)
3. 3.3 [Управление ветками](#)
4. 3.4 [Работа с ветками](#)
5. 3.5 [Удалённые ветки](#)
6. 3.6 [Перебазирование](#)
7. 3.7 [Заключение](#)

4. [4. Git на сервере](#)

1. 4.1 [Протоколы](#)
2. 4.2 [Установка Git на сервер](#)
3. 4.3 [Генерация открытого SSH ключа](#)
4. 4.4 [Настраиваем сервер](#)
5. 4.5 [Git-демон](#)
6. 4.6 [Умный HTTP](#)
7. 4.7 [GitWeb](#)
8. 4.8 [GitLab](#)
9. 4.9 [Git-хостинг](#)
10. 4.10 [Заключение](#)

5. **Распределённый Git**

1. 5.1 [Распределённый рабочий процесс](#)
2. 5.2 [Участие в проекте](#)
3. 5.3 [Сопровождение проекта](#)
4. 5.4 [Заключение](#)

1. **6. GitHub**

1. 6.1 [Настройка и конфигурация учетной записи](#)
2. 6.2 [Внесение собственного вклада в проекты](#)
3. 6.3 [Сопровождение проекта](#)
4. 6.4 [Управление организацией](#)
5. 6.5 [Создание сценариев GitHub](#)
6. 6.6 [Заключение](#)

2. **7. Инструменты Git**

1. 7.1 [Выбор ревизии](#)
2. 7.2 [Интерактивное индексирование](#)
3. 7.3 [Припрятывание и очистка](#)
4. 7.4 [Подпись](#)
5. 7.5 [Поиск](#)
6. 7.6 [Перезапись истории](#)
7. 7.7 [Раскрытие тайн reset](#)
8. 7.8 [Продвинутое слияние](#)
9. 7.9 [Rerege](#)
10. 7.10 [Обнаружение ошибок с помощью Git](#)
11. 7.11 [Подмодули](#)
12. 7.12 [Создание пакетов](#)
13. 7.13 [Замена](#)
14. 7.14 [Хранилище учётных данных](#)
15. 7.15 [Заключение](#)

3. **8. Настройка Git**

1. 8.1 [Конфигурация Git](#)
2. 8.2 [Атрибуты Git](#)
3. 8.3 [Хуки в Git](#)
4. 8.4 [Пример принудительной политики Git](#)
5. 8.5 [Заключение](#)

4. **9. Git и другие системы контроля версий**

1. 9.1 [Git как клиент](#)
2. 9.2 [Переход на Git](#)
3. 9.3 [Заключение](#)

5. **10. Git изнутри**

1. 10.1 [Сантехника и Фарфор](#)
2. 10.2 [Объекты Git](#)
3. 10.3 [Ссылки в Git](#)
4. 10.4 [Pack-файлы](#)
5. 10.5 [Спецификации ссылок](#)
6. 10.6 [Протоколы передачи данных](#)
7. 10.7 [Обслуживание репозитория и восстановление данных](#)
8. 10.8 [Переменные окружения](#)
9. 10.9 [Заключение](#)

1. **A1. Приложение А: Git в других окружениях**

1. A1.1 [Графические интерфейсы](#)
2. A1.2 [Git в Visual Studio](#)
3. A1.3 [Git в Visual Studio Code](#)
4. A1.4 [Git в Eclipse](#)
5. A1.5 [Git в IntelliJ / PyCharm / WebStorm / PhpStorm / RubyMine](#)
6. A1.6 [Git в Sublime Text](#)
7. A1.7 [Git в Bash](#)
8. A1.8 [Git в Zsh](#)
9. A1.9 [Git в PowerShell](#)
10. A1.10 [Заключение](#)

2. **A2. Приложение В: Встраивание Git в ваши приложения**

1. A2.1 [Git из командной строки](#)
2. A2.2 [Libgit2](#)
3. A2.3 [JGit](#)
4. A2.4 [go-git](#)
5. A2.5 [Dulwich](#)

3. **A3. Приложение С: Команды Git**

1. A3.1 [Настройка и конфигурация](#)
2. A3.2 [Клонирование и создание репозитория](#)
3. A3.3 [Основные команды](#)
4. A3.4 [Ветвление и слияния](#)
5. A3.5 [Совместная работа и обновление проектов](#)
6. A3.6 [Осмотр и сравнение](#)
7. A3.7 [Отладка](#)
8. A3.8 [Внесение исправлений](#)
9. A3.9 [Работа с помощью электронной почты](#)
10. A3.10 [Внешние системы](#)
11. A3.11 [Администрирование](#)
12. A3.12 [Низкоуровневые команды](#)

2nd Edition

6.2 GitHub - Внесение собственного вклада в проекты

Внесение собственного вклада в проекты

Теперь наша учётная запись создана и настроена, давайте же пройдёмся по деталям, которые будут полезны при внесении вклада в уже существующие проекты.

Создание ответвлений (fork)

Если вы хотите вносить свой вклад в уже существующие проекты, в которых у нас нет прав на внесения изменений путём отправки (push) изменений, вы можете создать своё собственное ответвление (fork) проекта. Это означает, что GitHub создаст вашу собственную копию проекта, данная копия будет находиться в вашем пространстве имён и вы сможете легко делать изменения путём отправки (push) изменений.

Исторически так сложилось, что англоязычный термин «fork» (создание ветвления проекта) имел негативный контекстный смысл, данный термин означал, что кто-то повёл или ведёт проект с открытым исходным кодом в другом, отличном от оригинала, направлении, иногда данный термин так же означал создание конкурирующего проекта с отдельными авторами. В контексте GitHub, «fork» (создание ветвления проекта) просто означает создание ветвления проекта в собственном пространстве имён, что позволяет вносить публичные изменения и делать свой собственный вклад в более открытом виде.

Примечание

Таким образом, проекты не обеспокоены тем, чтобы пользователи, которые хотели бы выступать в роли соавторов, имели право на внесение изменений путём их отправки (push). Люди просто могут создавать свои собственные ветвления (fork), вносить туда изменения, а затем отправлять свои внесённые изменения в оригинальный репозиторий проекта путём создания запроса на принятие изменений (Pull Request), сами же запросы на принятие изменений (Pull Request) будут описаны далее. Запрос на принятие изменений (Pull Request) откроет новую ветвь с обсуждением отправляемого кода, и автор оригинального проекта, а так же другие его участники, могут принимать участие в обсуждении предлагаемых изменений до тех пор, пока автор проекта не будет ими доволен, после чего автор проекта может добавить предлагаемые изменения в проект.

Для того, чтобы создать ответвление проекта, зайдите на страницу проекта и нажмите кнопку «Создать ответвление» («Fork»), которая расположена в правом верхнем углу.



Рисунок 88. Кнопка «Создать ответвление» («Fork»)

Через несколько секунд вы будете перенаправлены на собственную новую проектную страницу, содержащую вашу копию, в которой у вас есть права на запись.

Рабочий процесс с использованием GitHub

GitHub разработан с прицелом на определённый рабочий процесс с использованием запросов на слияния. Этот рабочий процесс хорошо подходит всем: и маленьким, сплочённым вокруг одного репозитория, командам; и крупным распределённым компаниям, и группам незнакомцев, сотрудничающих над проектом с сотней копий. Рабочий процесс GitHub основан на [тематических ветках](#), о которых мы говорили в главе [Ветвление в Git](#).

Вот как это обычно работает:

1. Создайте форк проекта.
2. Создайте тематическую ветку на основании ветки master.
3. Создайте один или несколько коммитов с изменениями, улучшающих проект.
4. Отправьте эту ветку в ваш проект на GitHub.
5. Откройте запрос на слияние на GitHub.
6. Обсуждайте его, вносите изменения, если нужно.
7. Владелец проекта принимает решение о принятии изменений, либо об их отклонении.
8. Получите обновлённую ветку master и отправьте её в свой форк.

Очень напоминает подход, описанный в разделе [Диспетчер интеграции](#) главы 5, но вместо использования электронной почты, команда сотрудничает через веб-интерфейс.

Давайте посмотрим, как можно предложить изменения в проект, размещённый на GitHub.

В большинстве случаев можно использовать официальный инструмент **GitHub CLI** вместо веб-Подсказка интерфейса GitHub. Инструмент доступен в системах Windows, MacOS и Linux. Посетите страницу [GitHub CLI homepage](https://github.com/cli/cli) для получения инструкций по установке и использованию.

Создание запроса на слияние

Тони ищет, чего бы запустить на своём новеньком Arduino. Кажется, он нашёл классный пример на <https://github.com/schacon/blink>.

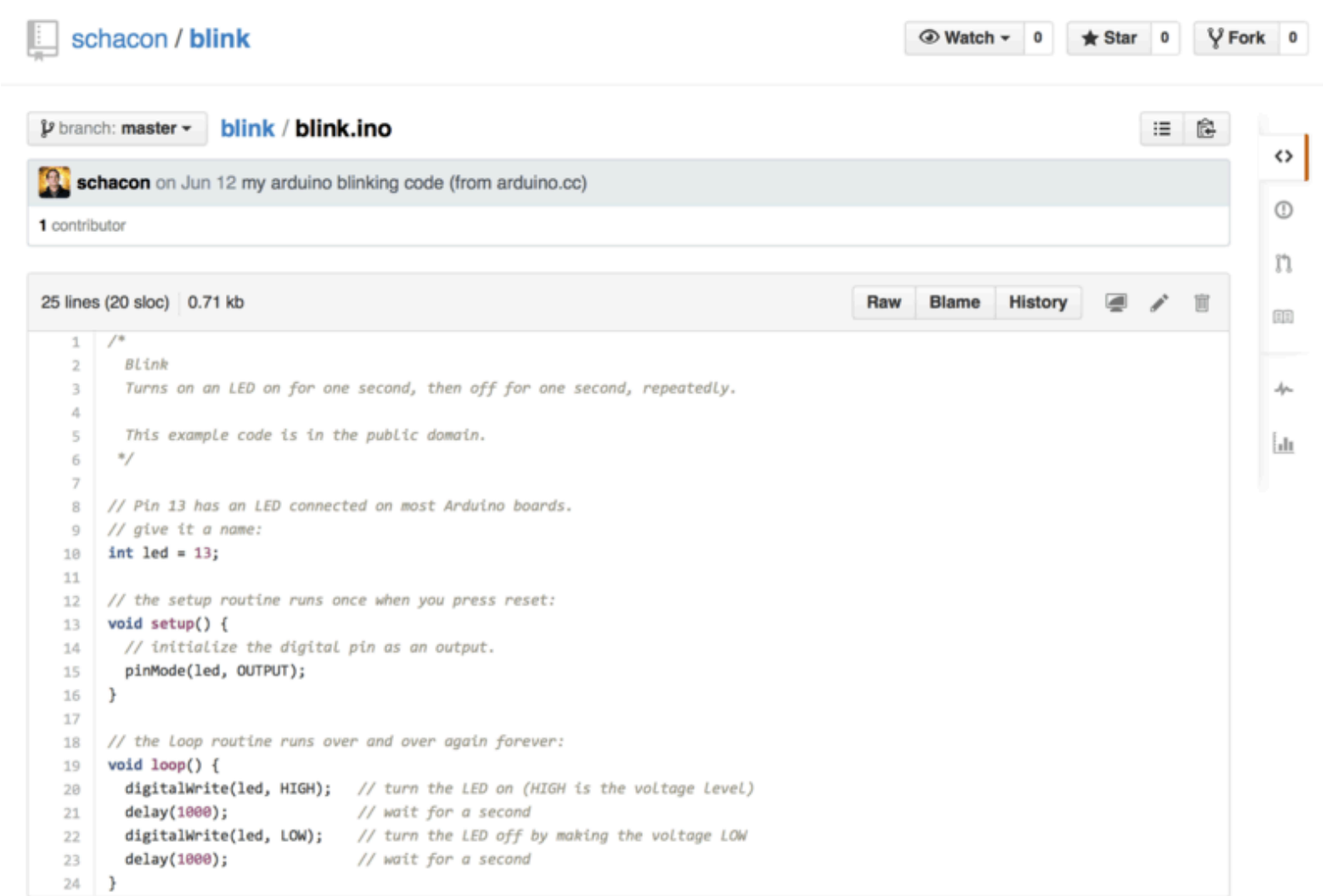


Рисунок 89. Проект, над которым мы хотим поработать

Единственная проблема в том, что светодиод моргает слишком быстро; нам кажется, лучше установить задержку в три секунды, не одну. Так давайте исправим это и предложим изменения автору.

Для начала, нажмите кнопку «Fork», как было сказано выше, чтобы заполнить собственную копию проекта. Мы зарегистрированы на GitHub под именем «tonychacon», так что наша копия окажется по адресу <https://github.com/tonychacon/blink>, где мы сможем редактировать её. Мы клонируем его, создадим тематическую ветку, внесём необходимые изменения и, наконец, отправим их на GitHub.

```
$ git clone https://github.com/tonychacon/blink (1)
Cloning into 'blink'...
```

```
$ cd blink
$ git checkout -b slow-blink (2)
Switched to a new branch 'slow-blink'
```

```
$ sed -i '' 's/1000/3000/' blink.ino (macOS) (3)
# If you're on a Linux system, do this instead:
# $ sed -i 's/1000/3000/' blink.ino (3)
```

```
$ git diff --word-diff (4)
diff --git a/blink.ino b/blink.ino
```

```

index 15b9911..a6cc5a5 100644
--- a/blink.ino
+++ b/blink.ino
@@ -18,7 +18,7 @@ void setup() {
// the loop routine runs over and over again forever:
void loop() {
    digitalWrite(led, HIGH);    // turn the LED on (HIGH is the voltage level)
    [-delay(1000);-]{+delay(3000);+}    // wait for a second
    digitalWrite(led, LOW);     // turn the LED off by making the voltage LOW
    [-delay(1000);-]{+delay(3000);+}    // wait for a second
}

$ git commit -a -m 'Change delay to 3 seconds' (5)
[slow-blink 5ca509d] Change delay to 3 seconds
1 file changed, 2 insertions(+), 2 deletions(-)

$ git push origin slow-blink (6)
Username for 'https://github.com': tonychacon
Password for 'https://tonychacon@github.com':
Counting objects: 5, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 340 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
To https://github.com/tonychacon/blink
 * [new branch]      slow-blink -> slow-blink

```

1. Клонировем нашу копию
2. Создаём тематическую ветку
3. Вносим свои изменения
4. Проверяем изменения
5. Фиксируем изменения в тематической ветку
6. Отправляем новую ветку в нашу копию на GitHub

Теперь, если мы зайдём на страничку нашей копии на GitHub, мы увидим, что GitHub заметил наши изменения и предлагает открыть запрос на слияние с помощью большой зелёной кнопки.

Также можно зайти на страницу «Branches», по адресу <https://github.com/<user>/<project>/branches>, найти интересующую ветку и открыть запрос оттуда.

Example file to blink the LED on an Arduino — Edit

2 commits

2 branches

0 releases

1 contributor

Your recently pushed branches:

slow-blink (less than a minute ago)

Compare & pull request



branch: master

blink / +



This branch is even with schacon:master

Pull Request

Compare

Create README.md



schacon authored on Jun 12

latest commit bbc80f9b29



README.md

Create README.md

4 months ago



blink.ino

my arduino blinking code (from arduino.cc)

4 months ago

README.md

Blink

This repository has an example file to blink the LED on an Arduino board.

<> Code

Pull Requests 0

Wiki

Pulse

Graphs

Settings

HTTPS clone URL

https://github.com/1

You can clone with HTTPS, SSH, or Subversion.

Clone in Desktop

Download ZIP

Рисунок 90. Кнопка открытия запроса на слияние

Если нажать на эту кнопку, появится экран ввода заголовка и описания предлагаемых изменений на рассмотрение владельцу проекта. Рекомендуется серьёзно подойти к составлению описания и сделать его максимально информативным, чтобы владелец проекта понимал, зачем эти изменения и какую пользу они принесут.

Также мы видим список коммитов в нашей тематической ветке, «опередивших» ветку master (в данном случае всего один коммит) и предпросмотр всех изменений, вносимых этими коммитами.

Рисунок 91. Страница создания запроса на слияние

После создания запроса на слияние (путём нажатия кнопки «Create pull request» на этой странице) владелец форкнутого проекта получит уведомление о предложенных изменениях со ссылкой на страницу с информацией о запросе.

Примечание Запросы на слияние широко используются для публичных проектов типа описанного выше, когда участник уже подготовил все изменения для слияния с основным репозиторием. Тем не менее, часто можно встретить использование запросов на слияние во внутренних проектах *в самом начале* цикла разработки. Объяснение простое: вы можете обновлять тематическую ветку **после** открытия запроса на слияние, поэтому сам запрос открывается как можно раньше чтобы отслеживать прогресс разработки.

Обработка запроса на слияние

На этом этапе, владелец проекта может просмотреть предложенные изменения, принять, отклонить или прокомментировать их. Предположим, ему импонирует идея, но он предпочёл бы большую задержку перед включением или выключением света.

В то время как в главе [Распределённый Git](#) обсуждение изменений может производиться через электронную почту, на GitHub всё происходит онлайн. Владелец проекта может просмотреть суммарные изменения, вносимые запросом, и прокомментировать любую отдельно взятую строку.

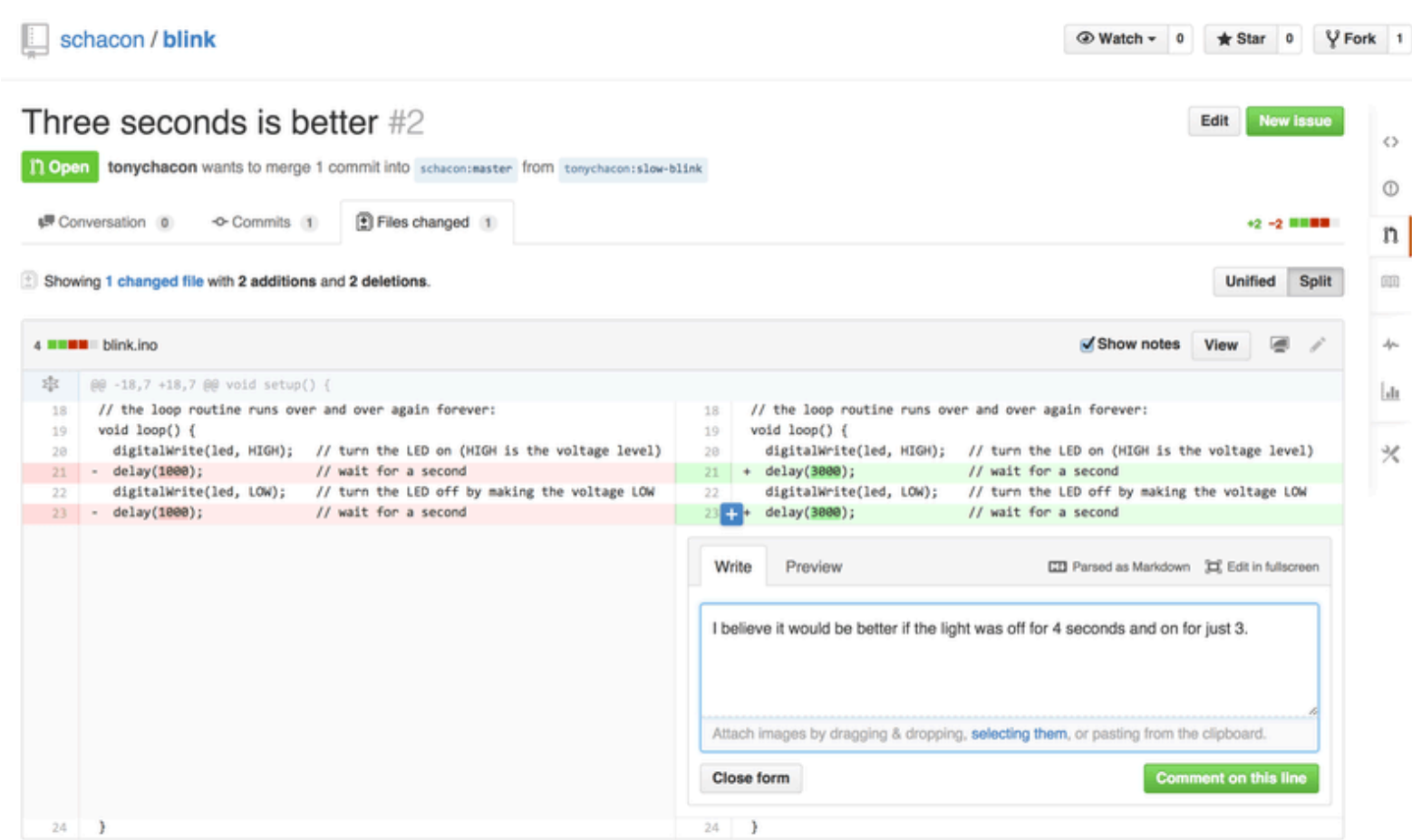


Рисунок 92. Комментирование определённой строки в запросе на слияние

Как только владелец прокомментирует изменения, автор запроса на слияние (а также все подписавшиеся на этот репозиторий) получают уведомления. Далее мы рассмотрим как настроить уведомления, но сейчас, если Тони включил уведомления через электронную почту, он получит следующее письмо:



Рисунок 93. Комментарии, отправленные по электронной почте

Кто угодно может оставлять комментарии к запросу на слияние. На [Страница обсуждения запроса на слияние](#) можно увидеть пример, где владелец проекта оставил комментарии как к строке кода, так и в основной секции обсуждения. Как вы могли заметить, комментарии к коду так же приведены в виде обсуждения.

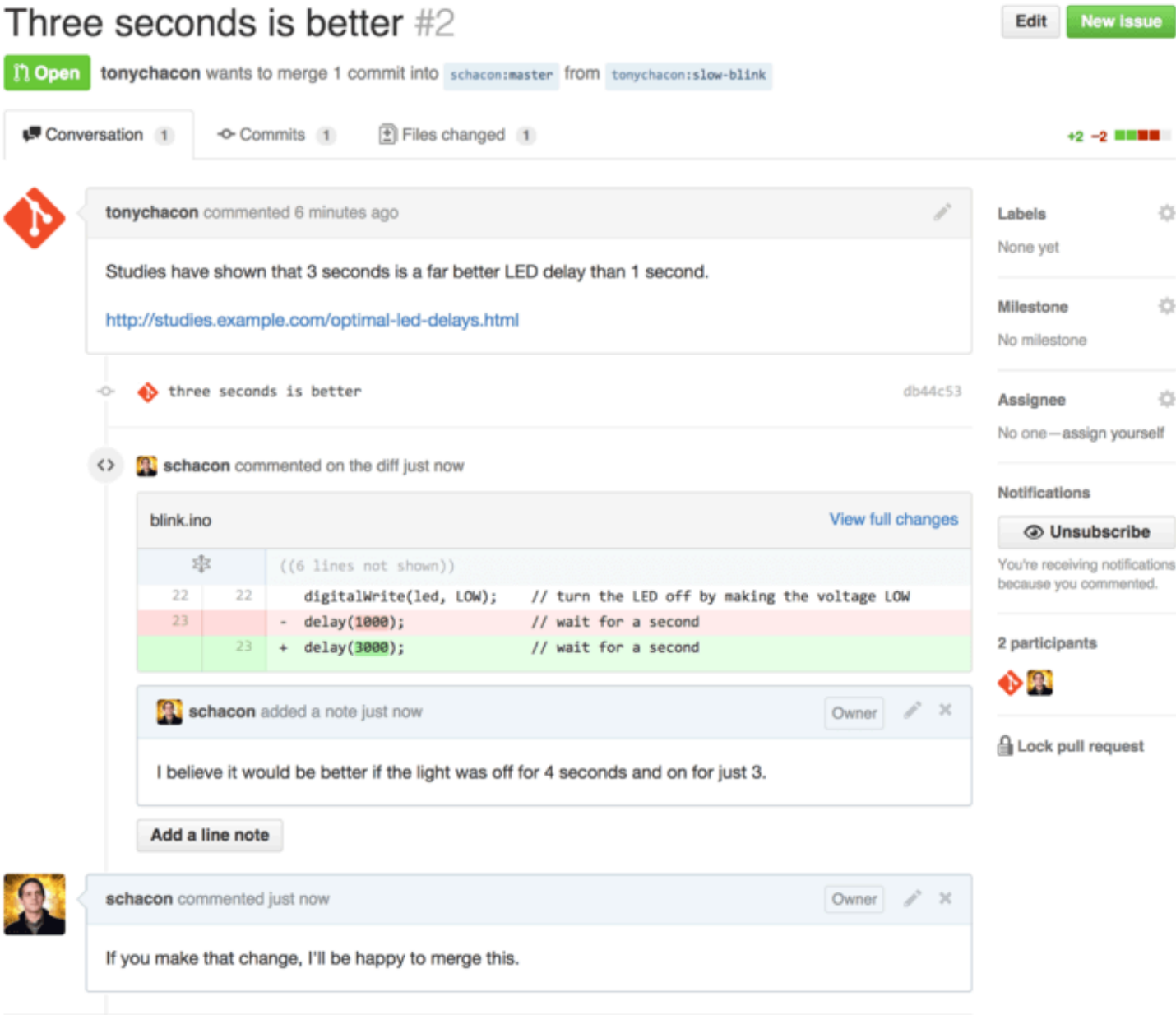


Рисунок 94. Страница обсуждения запроса на слияние

Теперь участник может видеть что ему необходимо сделать для того, чтобы его изменения были приняты. К счастью, это тоже легко сделать. Используя почту, вам потребуется заново отправить свои изменения в список рассылки, а при использовании GitHub вы просто делаете коммит в тематическую ветку и повторяете push, что автоматически обновляет запрос на слияние. На рисунке [Финальная стадия запроса на слияние](#) также видно, что в обновлённом запросе на слияние старый комментарий к коду был свёрнут, так как он относится к строке, которая с тех пор изменилась.

Когда участник сделает это, владелец проекта снова получит уведомление, а на странице запроса будет отмечено, что проблема решена. Фактически, как только строка кода имеющая комментарий будет изменена, GitHub заметит это и удалит устаревшее отличие.

Three seconds is better #2



tonychacon wants to merge 3 commits into `schacon:master` from `tonychacon:slow-blink`

Conversation 3

Commits 3

Files changed 1



tonychacon commented 11 minutes ago

Studies have shown that 3 seconds is a far better LED delay than 1 second.

<http://studies.example.com/optimal-led-delays.html>

three seconds is better

db44c53

schacon commented on an outdated diff 5 minutes ago

Show outdated diff



schacon commented 5 minutes ago

Owner

If you make that change, I'll be happy to merge this.

tonychacon added some commits 2 minutes ago

longer off time

8c1f66f

remove trailing whitespace

ef4725c



tonychacon commented 10 seconds ago

I changed it to 4 seconds and also removed some trailing whitespace that I found. Anything else you would like me to do?



This pull request can be automatically merged.

You can also merge branches on the [command line](#).



Merge pull request

Рисунок 95. Финальная стадия запроса на слияние

Примечательно, что если вы перейдёте на вкладку «Files Changed» в этом запросе на слияние, то увидите «унифицированную» разницу — это суммарные изменения, которые будут включены в основную ветку при слиянии тематической ветки. В терминологии `git diff` это эквивалентно команде `git diff master...<branch>` для ветки, на которой основан этот запрос на слияние. В [Определение применяемых изменений](#) детальнее описан данный тип отличий.

GitHub так же проверяет может ли запрос на слияние быть применён без конфликтов и предоставляет кнопку для осуществления слияния на сервере. Эта кнопка отображается только если у вас есть права на запись в репозиторий и возможно простейшее слияние. По нажатию на неё GitHub произведёт «non-fast-forward» слияние, что значит даже если слияние **может** быть осуществлено перемоткой вперед, всё равно будет создан коммит слияния.

При желании, можно стянуть ветку и произвести слияние локально. Если эта ветка будет слита в `master` ветку и отправлена на сервер, то GitHub автоматически закроет запрос на слияние.

Это основной рабочий процесс, который используется большинством проектов на GitHub. Создаются тематические ветки, открываются запросы на слияние, производится обсуждение, при необходимости производятся доработки в ветке и, наконец, запрос либо закрывается, либо сливается.

Не только ответвления

Примечание

Важно отметить, что можно открывать запросы на слияние между двумя ветками в одном репозитории. Если вы работаете над функционалом с кем-то ещё и у вас обоих есть права записи, то вы можете отправить свою тематическую ветку в репозиторий и открыть запрос на слияние в master ветку в рамках одного проекта, что позволит инициировать процедуру проверки кода и его обсуждения. Создание ответвлений проекта не является обязательным.

Продвинутые запросы на слияние

На текущий момент мы рассмотрели основы участия в проекте на GitHub, давайте рассмотрим некоторые интересные секреты и уловки касательно запросов слияния, чтобы вы могли более эффективно их использовать.

Запросы слияния как Патчи

Важно понимать, что многие проекты не воспринимают запросы слияния как очередь идеальных патчей, которые должны применяться аккуратно и по порядку, как и большинство проектов, участие в которых основывается на отправке набора патчей через списки почтовых рассылок. Большинство проектов на GitHub понимают ветки запросов на слияние как беседу относительно предлагаемого изменения, завершающуюся слиянием унифицированных изменений.

Это важное различие, так как изменение предлагается до того, как код станет считаться идеальным, что гораздо реже происходит с распространяемыми наборами патчей через списки рассылок. Обсуждение происходит на более раннем этапе и выработка правильного решения происходит за счёт усилий сообщества. Когда код предлагается через запрос на слияние и сопровождающий проекта или сообщество предлагает изменения, то не применяется набор патчей, а отправляются результирующие изменения как новый коммит в ветку, двигая обсуждение вперёд и сохраняя уже проделанную работу нетронутой.

Например, если вы вернётесь и посмотрите на [Финальная стадия запроса на слияние](#), то увидите, что участник не делал перебазирование своего коммита и не отправлял новый запрос на слияние. Вместо этого были сделаны новые коммиты и отправлены в существующую ветку. Таким образом, если вы в будущем вернётесь к этому запросу слияния, то легко найдёте весь контекст принятого решения. По нажатию кнопки «Merge» целенаправленно создаётся коммит слияния, который указывает на запрос слияния, оставляя возможность возврата к цепочке обсуждения.

Следование за исходным репозиторием

Если ваш запрос на слияние устарел или не может быть слит без конфликтов, то вам нужно изменить его, чтобы сопровождающий мог просто его слить. GitHub проверит это за вас и под каждым из запросов на слияние отобразит уведомление, можно ли его слить без конфликтов или нет.

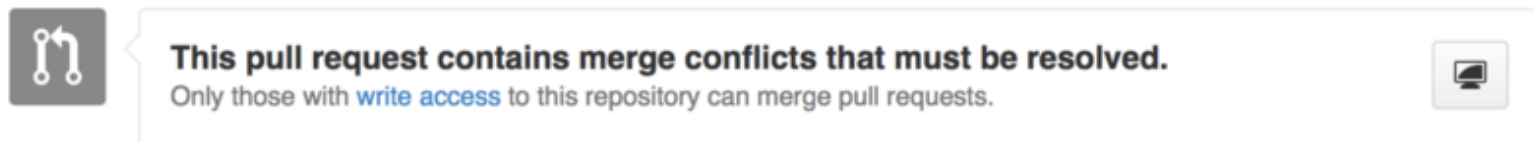


Рисунок 96. Запрос имеет конфликты слияния

Если вы видите что-то вроде [Запрос имеет конфликты слияния](#), то вам следует изменить свою ветку так, чтобы исключить конфликты и сопровождающий не делал лишнюю работу.

Существует два основных варианта это сделать. Вы можете либо перебазировать свою ветку относительно целевой ветки (обычно, относительно master ветки исходного репозитория), либо слить целевую ветку в свою.

Большинство разработчиков на GitHub выбирают последний вариант по тем же причинам, что и мы в предыдущем разделе. Важна история и окончательное слияние, а перебазирование не принесёт вам ничего, кроме немного более чистой истории, при этом оно **гораздо** сложнее и может стать источником ошибок.

Если вы хотите сделать запрос на слияние применяемым, то следует добавить исходный репозиторий как новый удалённый, слить изменения из его основной ветки в вашу тематическую, если имеются исправления, и, наконец, отправить все изменения в ту ветку, на основании которой был открыт запрос на слияние.

Предположим, что в примере «tonychacon», который мы использовали ранее, основной автор сделал изменения, которые конфликтуют с запросом на слияние. Рассмотрим это пошагово.

```
$ git remote add upstream https://github.com/schacon/blink (1)

$ git fetch upstream (2)
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (3/3), done.
Unpacking objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
From https://github.com/schacon/blink
 * [new branch]      master      -> upstream/master

$ git merge upstream/master (3)
Auto-merging blink.ino
CONFLICT (content): Merge conflict in blink.ino
Automatic merge failed; fix conflicts and then commit the result.

$ vim blink.ino (4)
$ git add blink.ino
$ git commit
[slow-blink 3c8d735] Merge remote-tracking branch 'upstream/master' \
    into slower-blink

$ git push origin slow-blink (5)
Counting objects: 6, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 682 bytes | 0 bytes/s, done.
Total 6 (delta 2), reused 0 (delta 0)
To https://github.com/tonychacon/blink
ef4725c..3c8d735  slower-blink -> slow-blink
```

- 1. Добавляем исходный репозиторий как удалённый с именем upstream.
- 2. Получаем последние изменения из него.
- 3. Сливаем основную ветку в нашу тематическую.
- 4. Исправляем указанный конфликт.
- 5. Отправляем изменения в ту же тематическую ветку.

Как только это будет сделано, запрос на слияние будет автоматически обновлён и перепроверен на возможность слияния.

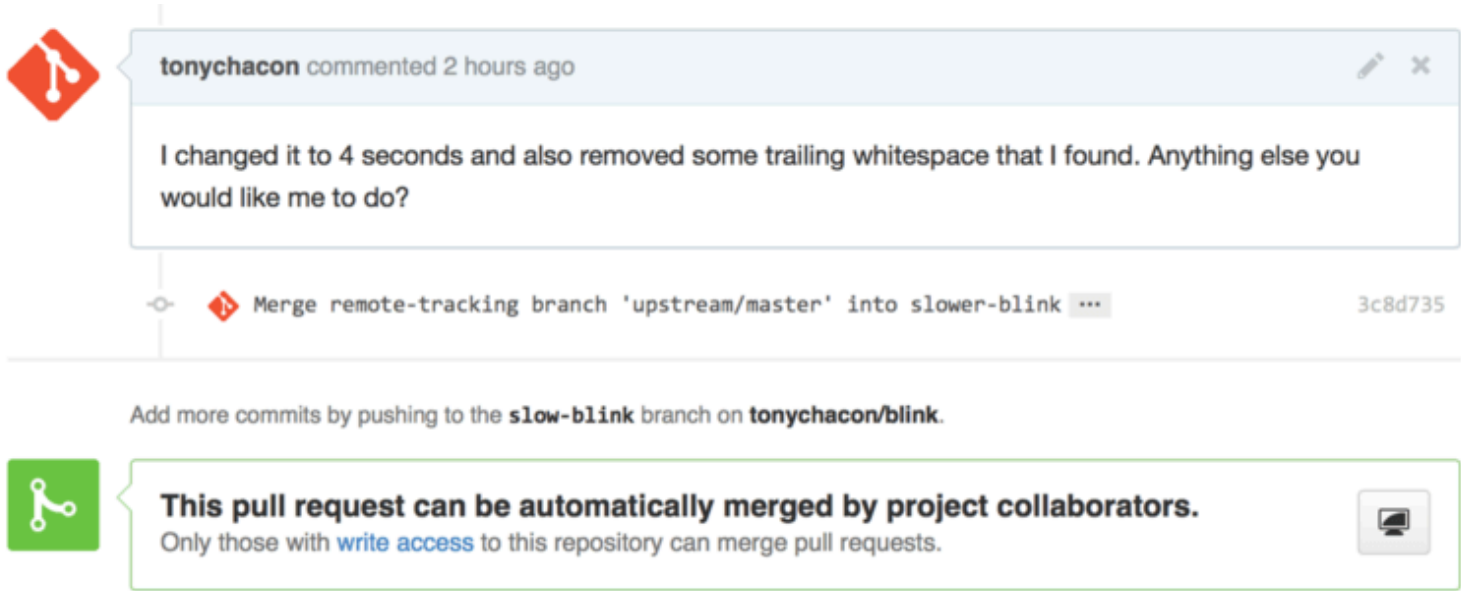


Рисунок 97. Запрос слияния без конфликтов

Одна из замечательных особенностей Git - это то, что вы можете делать это постоянно. Если у вас очень длительный проект, вы можете легко сливать изменения из целевой ветки снова и снова и иметь дело только с конфликтами, возникшими с момента вашего последнего слияния, что делает процесс очень управляемым.

Если вы очень хотите перебазировать ветку, чтобы её почистить, то, конечно, вы можете это сделать, но настоятельно не рекомендуется переписывать ветку, к которой уже открыт запрос на слияние. Если другие люди уже стянули её и проделали много работы, то вы столкнётесь со всеми проблемами, описанными в разделе [Опасности перемещения](#) главы 3. Вместо этого, отправьте перебазированную ветку в новую на GitHub и откройте новый запрос на слияние, который указывает на предыдущий, затем закройте исходный.

Ссылки

Возможно, ваш следующий вопрос будет: «Как мне сослаться на предыдущий запрос слияния?» Оказывается, существует много способов сослаться на другие вещи практически везде, где у вас есть права записи на GitHub.

Давайте начнём с перекрёстных ссылок для запросов слияния или проблем. Всем запросам слияния и проблемам присваиваются уникальные номера в пределах проекта. Например, у вас не может быть запроса на слияние с номером #3 и проблемы с номером #3. Если вы хотите сослаться на любой запрос слияния или проблему из другого места, просто добавьте #<num> в комментарий или описание. Так же можно указывать более конкретно, если проблема или запрос слияния находятся где-то ещё; пишите username#<num> если ссылаетесь на проблему или запрос слияния, находящиеся в ответвлённом репозитории, или username/repo#<num> если ссылаетесь на другой репозиторий.

Рассмотрим это на примере. Предположим, что мы перебазировали ветку в предыдущем примере, создали новый запрос слияния для неё и сейчас хотим сослаться на предыдущий запрос слияния из нового. Так же мы хотим сослаться на проблему, находящуюся в ответвлённом репозитории, и на проблему из совершенно другого проекта. Мы можем составить описание как указано на [Перекрёстные ссылки в запросе слияния](#).

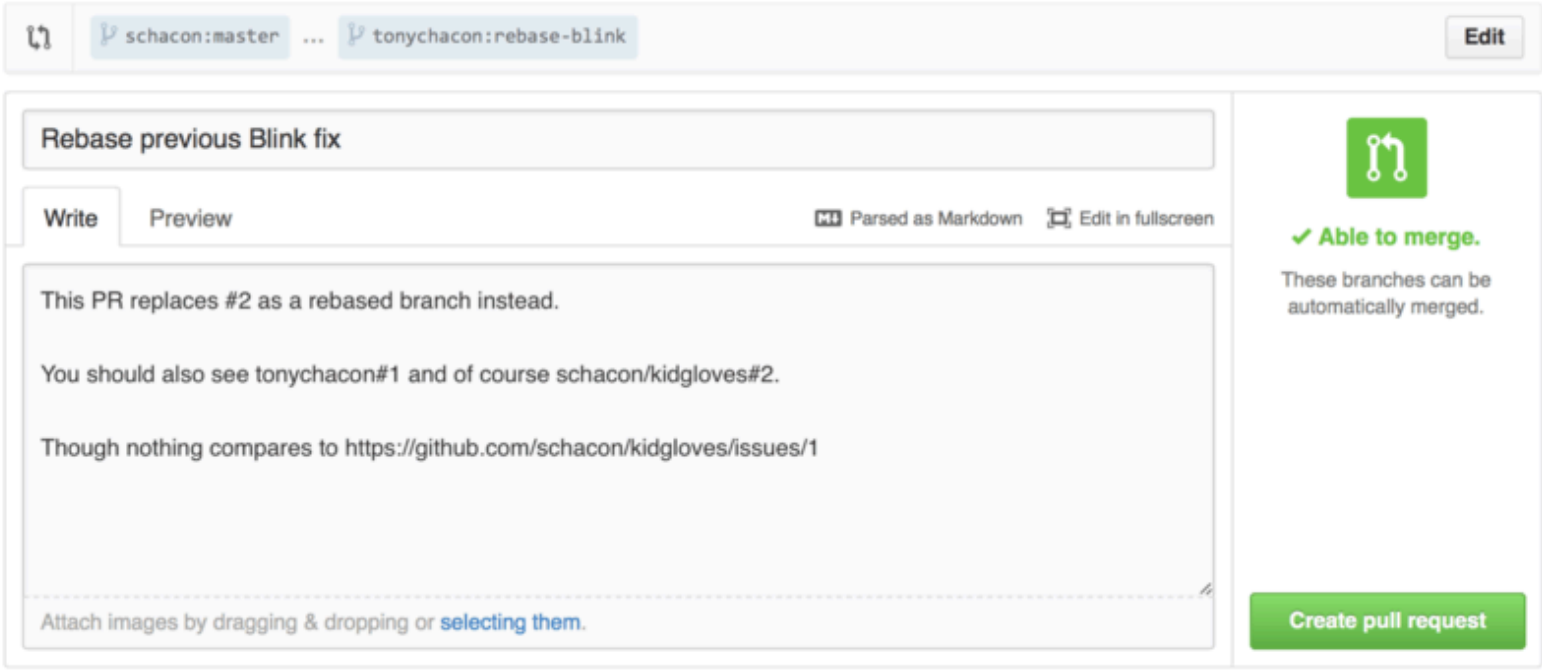


Рисунок 98. Перекрёстные ссылки в запросе слияния

Когда мы отправим запрос на слияние, то увидим что-то вроде [Отображение перекрёстных ссылок в запросе слияния](#).

Rebase previous Blink fix #4


Open

tonychacon wants to merge 2 commits into `schacon:master` from `tonychacon:rebase-blink`

Conversation 0

Commits 2

Files changed 1




tonychacon commented just now


This PR replaces #2 as a rebased branch instead.


You should also see [tonychacon#1](#) and of course [schacon/kidgloves#2](#).

Though nothing compares to [schacon/kidgloves#1](#)



tonychacon added some commits 4 hours ago

 three seconds is better

 remove trailing whitespace


afe904a

a5a7751


Рисунок 99. Отображение перекрёстных ссылок в запросе слияния

Заметьте, что указанная полная ссылка на GitHub была сокращена до необходимого минимума.

Если Тони сейчас вернётся назад и закроет оригинальный запрос слияния, то мы это увидим, так как он упомянут в новом, а GitHub автоматически создаст отслеживающее событие в хронике запроса слияния. Это значит, что все, кто просматривает закрытый запрос слияния, могут легко перейти к запросу слияния, который его заменил. Ссылка будет выглядеть как указано на [Отображение перекрёстных ссылок в закрытом запросе слияния](#).

 Merge remote-tracking branch 'upstream/master' into slower-blink ...


3c8d735




tonychacon referenced this pull request 2 minutes ago

Rebase previous Blink fix #4

Open



tonychacon closed this just now



Closed with unmerged commits

This pull request is closed, but the `tonychacon:slow-blink` branch has unmerged commits.

Delete branch

Рисунок 100. Отображение перекрёстных ссылок в закрытом запросе слияния

Кроме идентификационных номеров, можно ссылаться на конкретный коммит используя SHA-1. Следует указывать полный 40 символьный хеш SHA-1, но если GitHub увидит его в комментарии, то автоматически подставит ссылку на коммит. Как было сказано выше, вы можете ссылаться на коммиты как в других, так и в ответвлённых репозиториях точно так же, как делали это с Проблемами.

GitHub-версия разметки Markdown

Ссылки на другие Проблемы — это лишь часть интереснейших вещей, которые вы можете делать в текстовых полях на GitHub. Для «проблемы» или «запроса слияния» в полях описания, комментария, комментария кода и

других вы можете использовать так называемую «GitHub-версию разметки Markdown». Разметка похожа на обычный текст, который основательно преобразуется.

Смотрите [Пример написания и отображения текста с разметкой](#) для примера как использовать разметку при написании комментариев и текста.

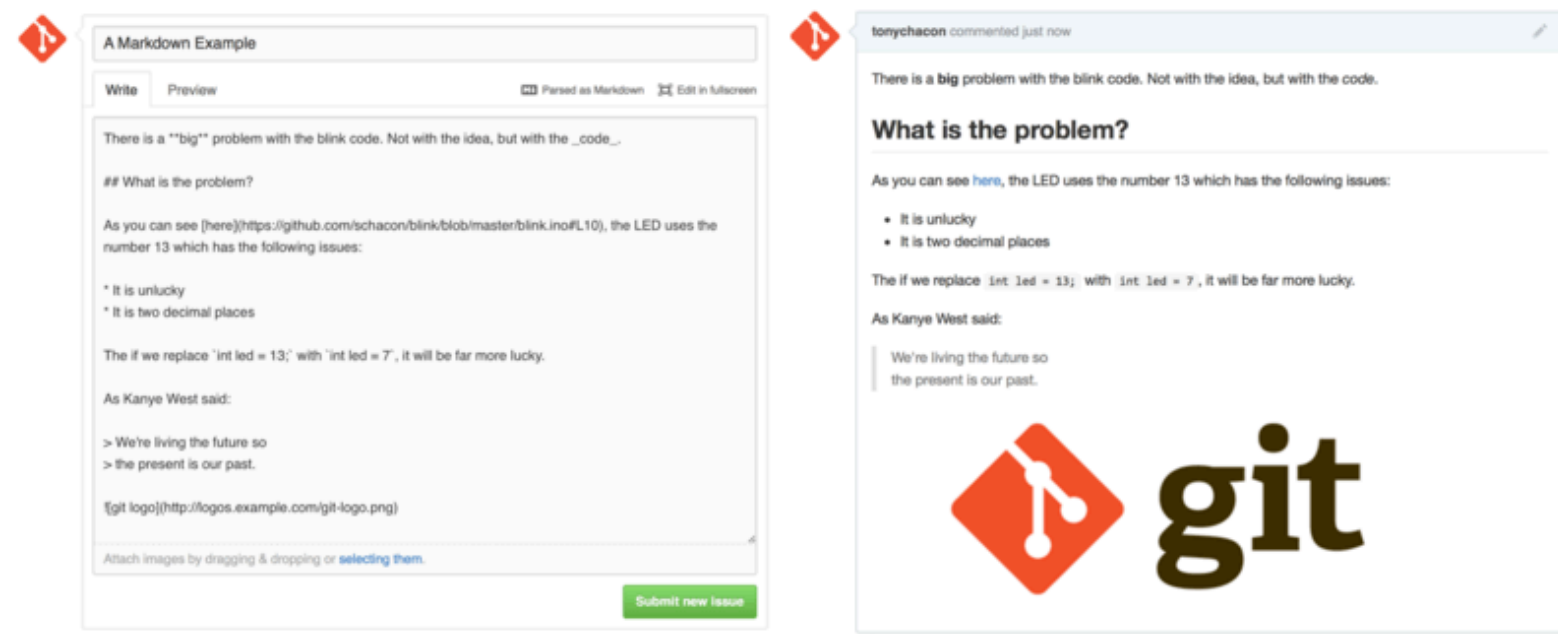


Рисунок 101. Пример написания и отображения текста с разметкой

GitHub расширил возможности обычной разметки. Эти возможности могут быть очень полезными при создании запросов слияния или комментариев и описаний к проблемам.

Списки задач

Список задач — это первая действительно важная возможность специфической разметки GitHub, особенно для запросов слияния. Список задач представляет собой список флажков для задач, которые вы хотите выполнить. Размещение его в описании Проблемы или запроса на слияние обычно указывает на то, что должно быть сделано до того, как проблема будет считаться решённой.

Список задач можно добавить следующим образом:

- [X] Write the code
- [] Write all the tests
- [] Document the code

Если добавить этот список в описание запроса на слияние или проблемы, то он будет отображён следующим образом [Отображение списка задач в комментарии](#)

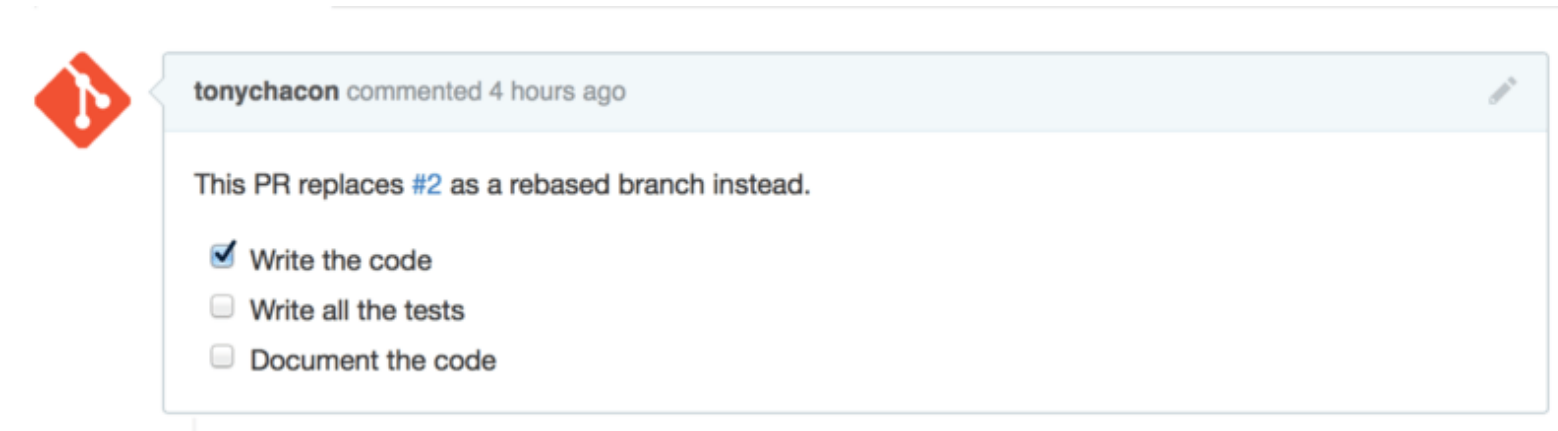


Рисунок 102. Отображение списка задач в комментарии

Он часто используется в запросах на слияние для отображения списка того, что вы хотите сделать до того, как запрос будет готов к слиянию. Вы можете просто кликнуть по флажку, чтобы обновить комментарий — не нужно редактировать комментарий вручную, чтобы пометить задачу как выполненную.

Так же GitHub ищет списки задач в запросах на слияние и проблемах и отображает их как метаданные на страницах, где они упоминаются. Например, если в вашем запросе на слияние есть задачи и вы просматриваете список всех запросов, то можно увидеть на сколько готов каждый из них. Это позволяет разбивать запрос на слияние на несколько подзадач и помогает другим людям отслеживать прогресс ветки. Пример приведён на [Статистика задач в списке запросов слияния](#).

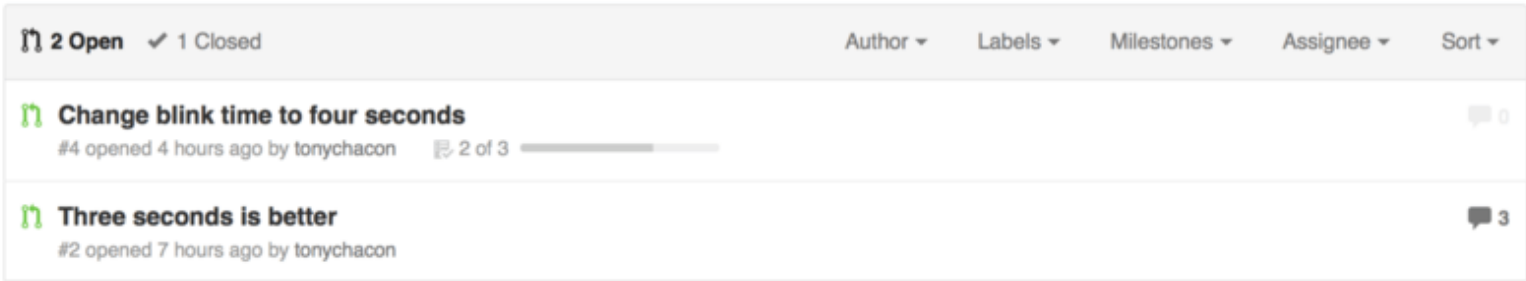


Рисунок 103. Статистика задач в списке запросов слияния

Такая возможность невероятно полезна когда вы открываете запрос на слияние на раннем этапе реализации и отслеживаете прогресс с помощью него.

Отрывки кода

В комментарии так же можно вставлять отрывки кода. Это особенно полезно когда вы хотите показать что-нибудь, что вы *собираетесь* попробовать сделать, до того, как включить это в вашу ветку. Так же часто применяется для добавления примеров кода, который не работает или мог быть добавлен в запрос на слияние.

Для добавления отрывка кода следует обрамить его обратными кавычками.

```
```java
for(int i=0 ; i < 5 ; i++)
{
 System.out.println("i is : " + i);
}
```
```

Если вы укажете название языка, как показано на примере, GitHub попыбует применить к нему подсветку синтаксиса. Для приведённого примера код будет выглядеть как на [Отображение обрамлённого кода](#).



Рисунок 104. Отображение обрамлённого кода

Цитирование

Если вы отвечаете только на часть большого комментария, то можно цитировать только выбранную часть, предваряя её символом >. Это настолько часто используется, что даже существует комбинация клавиш для этого. Если в комментарии выделить текст, на который вы собираетесь ответить, и нажать клавишу r, то выделенный текст будет включён как цитата в ваш комментарий.

Цитаты выглядят примерно так:

```
> Whether 'tis Nobler in the mind to suffer
> The Slings and Arrows of outrageous Fortune,
```

How big are these slings and in particular, these arrows?

После обработки комментарий будет выглядеть как [Пример отображения цитаты](#).

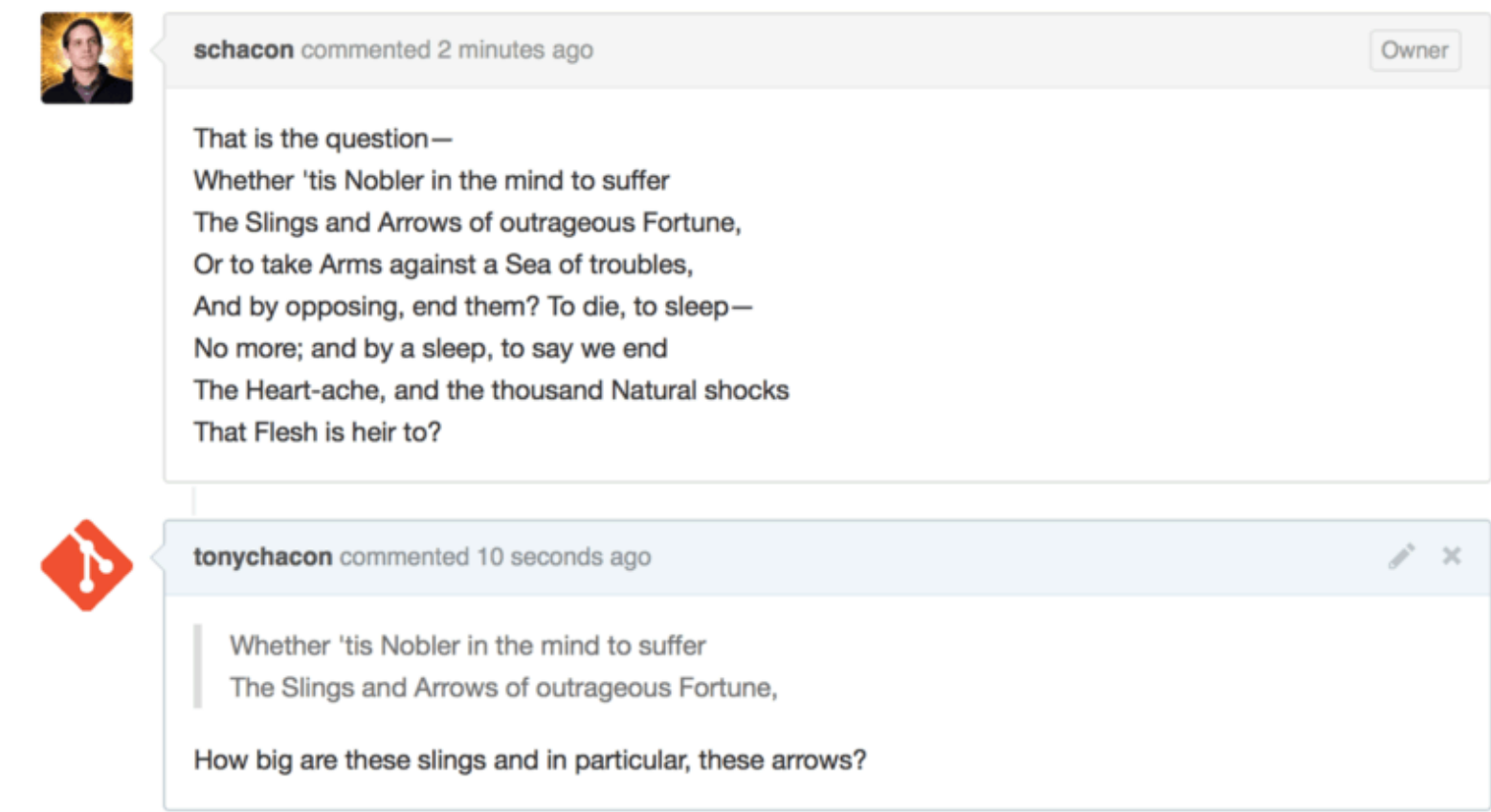


Рисунок 105. Пример отображения цитаты

Смайлики

Наконец, вы можете использовать смайлики. На GitHub вы можете часто встретить их в комментариях или запросах на слияние. Для них есть даже помощник. Например, если при наборе комментария ввести символ двоеточия :, то будут предложены варианты автодополнения.

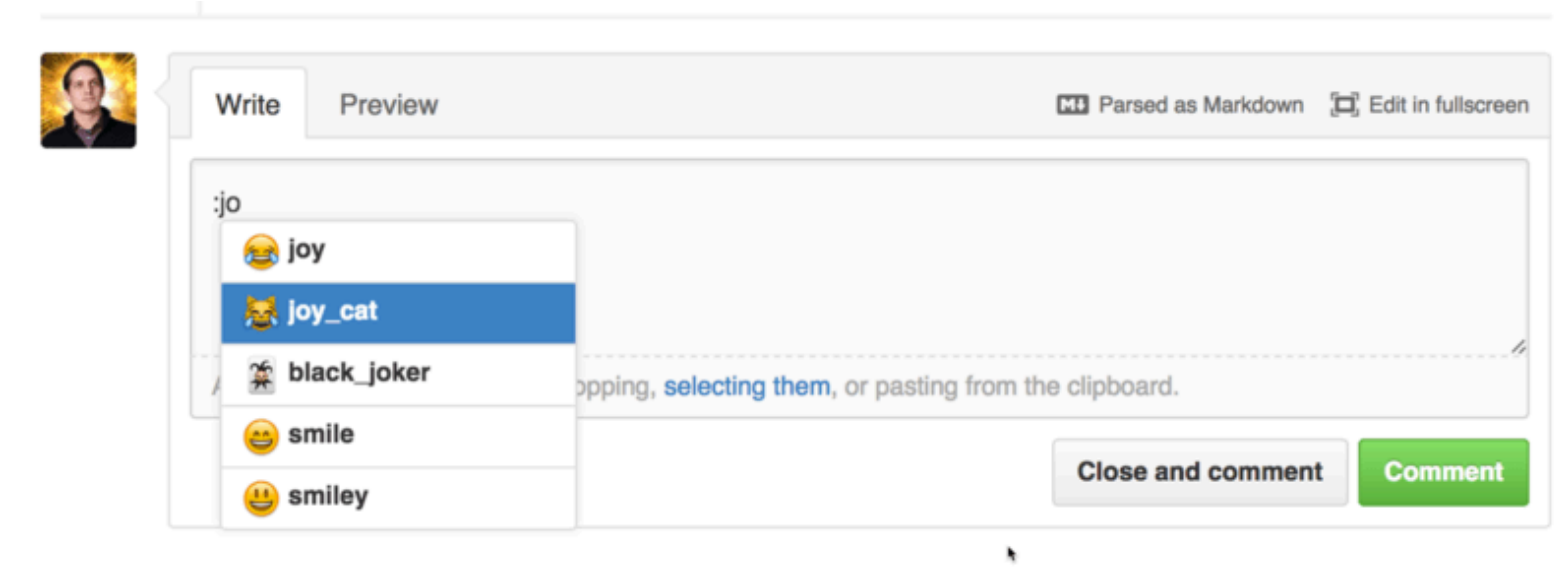


Рисунок 106. Автодополнение для смайлов в действии

Смайлы имеют вид :<name>: и могут располагаться в любом месте комментария. Например, вы можете написать что-нибудь вроде этого:

I :eyes: that :bug: and I :cold_sweat:.
:trophy: for :microscope: it.
:+1: and :sparkles: on this :ship:, it's :fire::poop:!

:clap::tada::panda_face:

Такой комментарий будет выглядеть как на [Перегруженный смайликами комментарий](#).

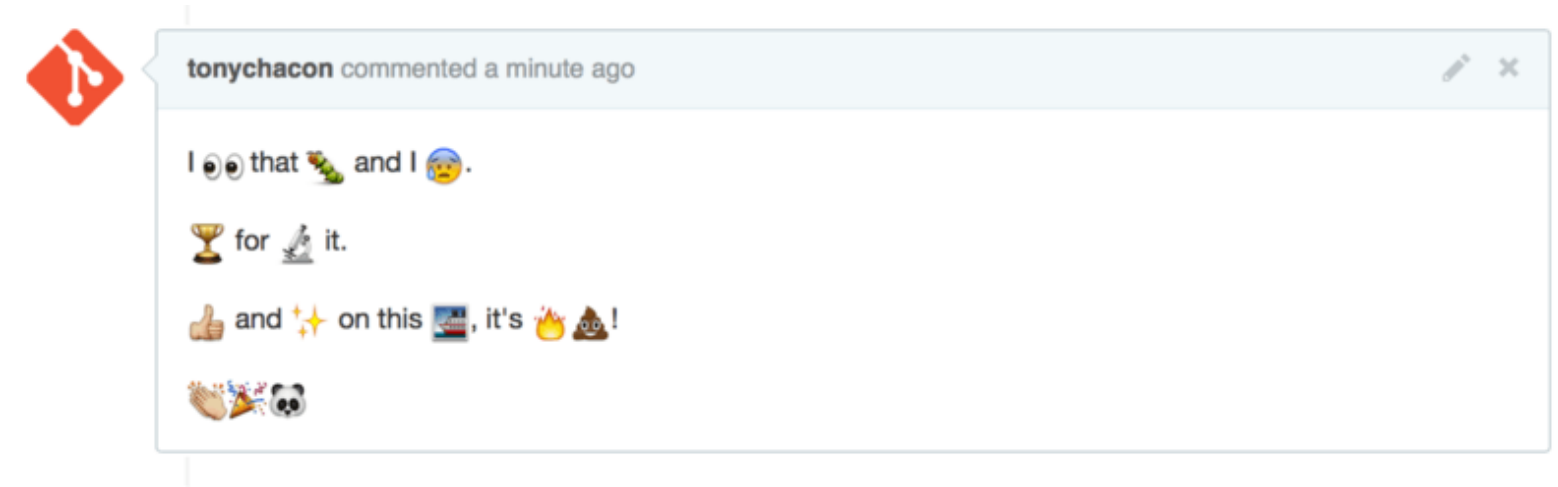


Рисунок 107. Перегруженный смайликами комментарий

Не то чтобы это невероятно полезно, но добавляет немного веселья и эмоций там, где трудно выразить какие-то эмоции.

Примечание На текущий момент существует много интернет сервисов, где используются смайлики. Отличную шпаргалку по поиску смайликов, которые выражают нужную вам эмоцию, можно найти здесь: <https://www.webfx.com/tools/emoji-cheat-sheet/>

Картинки

Технически, картинки не относятся к разметке GitHub, но их использование очень полезно. В дополнение к ссылкам на картинки в комментариях, GitHub позволяет встраивать картинки в комментарии.

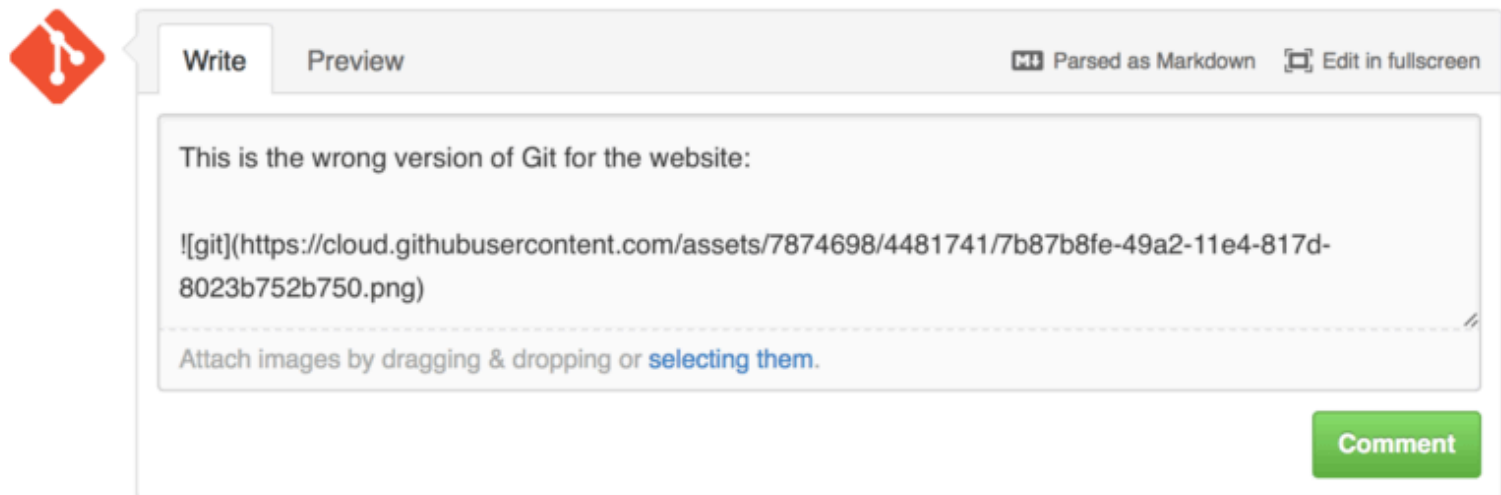
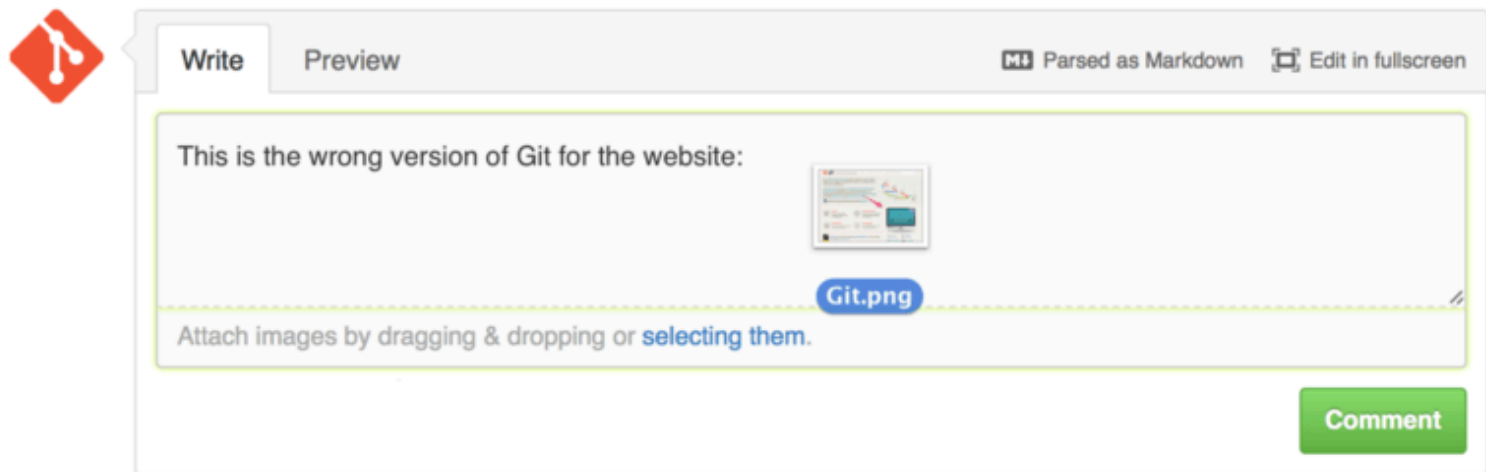


Рисунок 108. Перетаскивание картинке для загрузки и встраивания

Если вернуться немного назад к [Перекрёстные ссылки в запросе слияния](#), то над областью редактирования вы увидите небольшую подсказку «Parsed as Markdown». Нажав на неё, вы получите полную подсказку по использованию GitHub разметки.

Поддержание GitHub репозитория в актуальном состоянии

После создания форка, ваш репозиторий будет существовать независимо от оригинального репозитория. В частности, при появлении в оригинальном репозитории новых коммитов GitHub информирует вас следующим сообщением:

```
This branch is 5 commits behind progit:master.
```

При этом GitHub никогда не обновляет ваш репозиторий — это вы должны делать сами. К счастью, это очень просто сделать.

Первый способ не требует конфигурации. Например, если вы сделали форк репозитория <https://github.com/progit/progit2.git>, то актуализировать ветку master можно следующим образом:

```
$ git checkout master (1)
$ git pull https://github.com/progit/progit2.git (2)
$ git push origin master (3)
```

1. Если вы находитесь на другой ветке — перейти на ветку master.
2. Получить изменения из репозитория <https://github.com/progit/progit2.git> и слить их с веткой master.
3. Отправить локальную ветку master в ваш форк origin.

Каждый раз писать URL репозитория для получения изменений достаточно утомительно. Этот процесс можно автоматизировать слегка изменив настройки:

```
$ git remote add progit https://github.com/progit/progit2.git (1)
$ git fetch progit (2)
$ git branch --set-upstream-to=progit/master master (3)
$ git config --local remote.pushDefault origin (4)
```

1. Добавить исходный репозиторий как удалённый и назвать его progit.
2. Получить ветки репозитория progit, в частности ветку master.
3. Настроить локальную ветку master на получение изменений из репозитория progit.
4. Установить origin как репозиторий по умолчанию для отправки.

После этого, процесс обновления становится гораздо проще:

```
$ git checkout master (1)
$ git pull (2)
$ git push (3)
```

1. Если вы находитесь на другой ветке — перейти на ветку master.
2. Получить изменения из репозитория progit и слить их с веткой master.
3. Отправить локальную ветку master в ваш форк origin.

Данный подход не лишён недостатков. Git будет молча выполнять указанные действия и не предупредит вас в случае, когда вы добавили коммит в master, получили изменения из progit и отправили всё вместе в origin — все эти операции абсолютно корректны. Поэтому вам стоит исключить прямое добавление коммитов в ветку master, поскольку эта ветка фактически принадлежит другому репозиторию.

[prev](#) | [next](#)
[About this site](#)

Patches, suggestions, and comments are welcome.
Git is a member of [Software Freedom Conservancy](#).

