Это руководство научит вас делать изменения в проекте на GitHub. Описываемый процесс предлагает лучшие практики, и является достаточно распространённым — вы сможете применять его за пределами нашего сообщества. В проектах сообщества придерживаться такого процесса очень рекомендуется.

Сначала мы приведём высокоуровневое описание процесса, а затем подробно опишем каждый этап.

Предполагается знание основ системы контроля версий Git. Если вы ещё не работали с Git, мы дадим ссылки на официальную русскоязычную документацию по необходимым командам.

Вам также потребуется аккаунт на GitHub. [Регистрация](https://github.com/join?source=header-home) бесплатная и требует указания лишь имени пользователя и электронной почты.

Вот процесс с высоты птичьего полёта.

1. Форкните проект.
2. Склонируйте репозиторий.
3. Создайте ветку для своей работы.
4. Сделайте необходимые изменения в файлах — коде, документации, тестах. Закоммитьте их в только что созданную ветку.
5. Убедитесь, что проект работает после ваших изменений.
6. Сделайте Pull Request.
7. Обсудите его с рецензентом в процессе Code Review. При необходимости, внесите изменения в свой Pull Request.
8. Когда все довольны, Pull Request принимают — с этого момента ваши изменения попали в исходный репозиторий (upstream) и являются частью проекта.

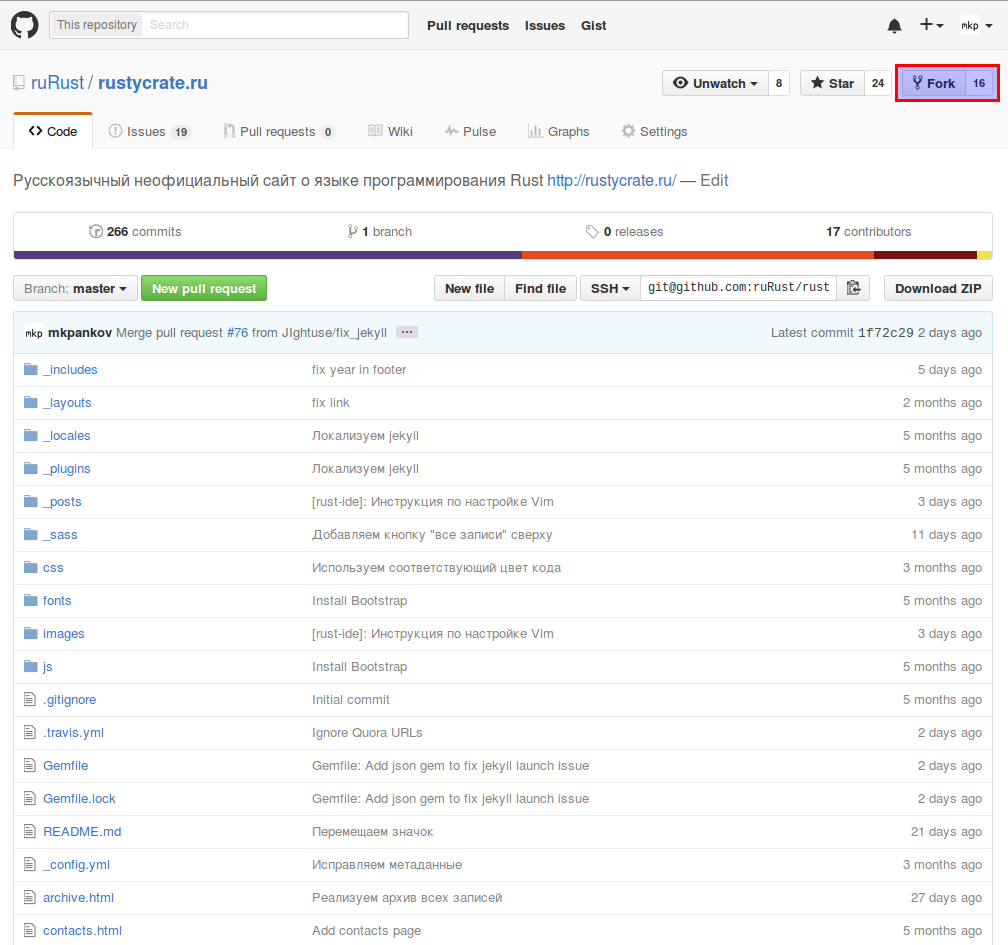
Работа над задачей закончена!

Теперь рассмотрим каждый этап подробнее.

Форкаем проект

Вы не можете отправлять коммиты (git push) напрямую в исходный репозиторий. По желанию хозяин проекта может это разрешить, но обычно доступ на запись есть только у людей, поддерживающих проект, а все остальные работают через Pull Request’ы («запросы на вливание изменений»; о них — ниже).

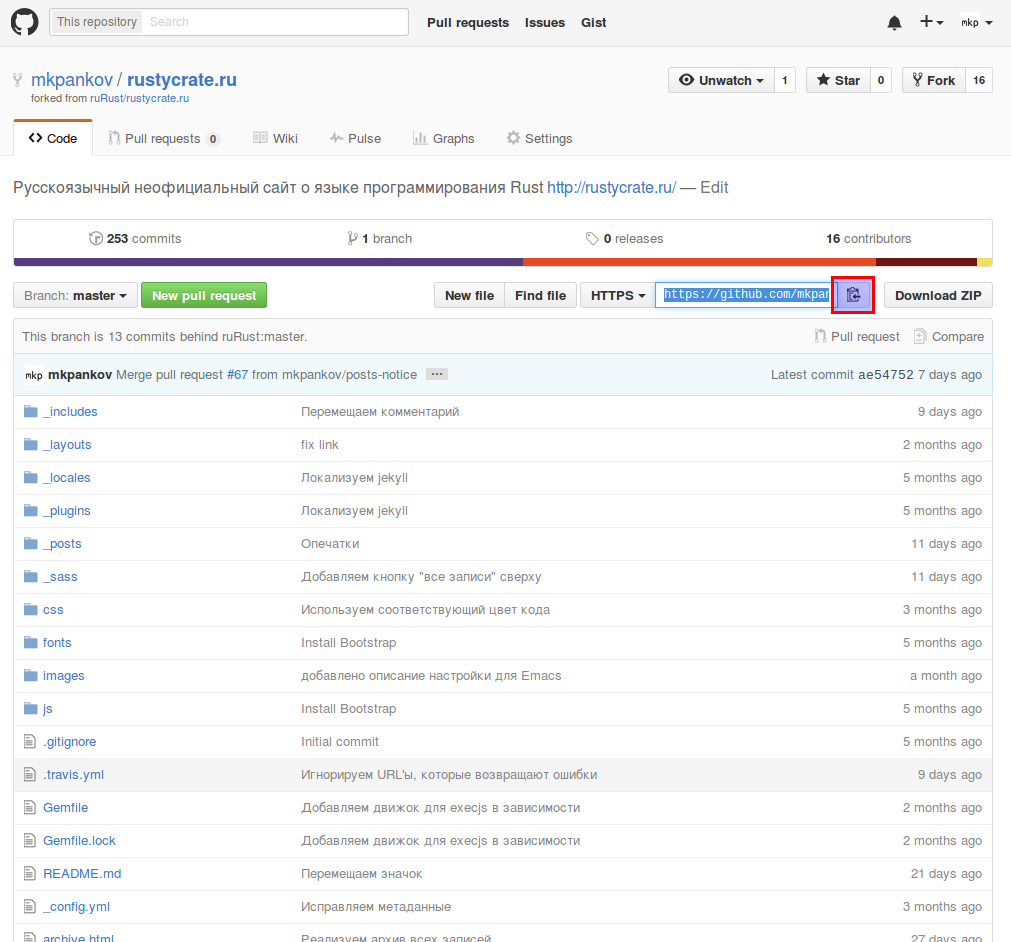
Поэтому мы форкаем проект — это создаст копию репозитория в вашем аккаунте. При этом у вас появится доступ на запись в вашу копию.



Через мгновение вы будете перенаправлены на страницу вашего форка.

Клонируем репозиторий

Затем нужно [склонировать репозиторий](https://git-scm.com/book/ru/v1/%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-Git-%D0%A1%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-Git-%D1%80%D0%B5%D0%BF%D0%BE%D0%B7%D0%B8%D1%82%D0%BE%D1%80%D0%B8%D1%8F" \l "%D0%9A%D0%BB%D0%BE%D0%BD%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5-%D1%81%D1%83%D1%89%D0%B5%D1%81%D1%82%D0%B2%D1%83%D1%8E%D1%89%D0%B5%D0%B3%D0%BE-%D1%80%D0%B5%D0%BF%D0%BE%D0%B7%D0%B8%D1%82%D0%BE%D1%80%D0%B8%D1%8F) на вашу локальную машину. Для этого нам нужен URL репозитория. Нажав на кнопку справа, вы скопируете его в буфер обмена. Обратите внимание на выбранный слева протокол. Если вы не настраивали SSH для GitHub, там должно быть указано HTTPS.



Затем выполняем команду в терминале (или командной строке Windows):

git clone <вставляем\_URL>

Репозиторий склонируется в под-директорию текущей директории. Например, если репозиторий называется foobar, у вас появится каталог foobar.

Создаём ветку

[Ветка](https://git-scm.com/book/ru/v1/%D0%92%D0%B5%D1%82%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D0%B2-Git-%D0%A7%D1%82%D0%BE-%D1%82%D0%B0%D0%BA%D0%BE%D0%B5-%D0%B2%D0%B5%D1%82%D0%BA%D0%B0%3F) по умолчанию — master. Чтобы изменениями было проще управлять и они не смешивались друг с другом, создадим отдельную ветку, где и будем работать. При этом ветку стоит назвать так, чтобы имя говорило о её назначении.

Например, мы хотим исправить ошибку в реализации протокола PROTOBAZ, так что наша ветка будет называться fix-protobaz.

Теперь заходим в наш склонированный репозиторий и [создаём ветку](https://git-scm.com/book/ru/v1/%D0%92%D0%B5%D1%82%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D0%B2-Git-%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-%D0%B2%D0%B5%D1%82%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F-%D0%B8-%D1%81%D0%BB%D0%B8%D1%8F%D0%BD%D0%B8%D1%8F):

|  |  |
| --- | --- |
| 1  2 | cd foobar  git checkout -b fix-protobaz |

Вторая команда создаст ветку и перейдёт на неё (сделает checkout).

Если после этого выполнить git status, он покажет

|  |  |
| --- | --- |
| 1  2 | On branch fix-protobaz  nothing to commit, working directory clean |

Эту команду стоит запомнить — когда не понимаете, в каком состоянии репозиторий, просто выполните её. Чаще всего в её выводе git покажет другие команды, которые делают то, что вы (скорее всего) и хотите сделать.

Делаем изменения

Теперь приступаем к работе. Редактируем код, обновляем документацию, чиним тесты, дополняем README.

Эти изменения мы коммитим в нашу ветку. Как это сделать — ниже.

При этом старайтесь делать коммиты часто, а сами коммиты — небольшими по объёму. Каждый коммит должен делать ровно одну вещь, и при этом поддерживать работоспособность проекта. Стремиться нужно к тому, чтобы в будущем можно было перейти на любой коммит и получить рабочий проект.

Если у вас сразу не получается придерживаться такой дисциплины, или изменения затрагивают весь проект «насквозь», допустимо ломать проект и постепенно чинить его в следующих коммитах.

Если вы уже достаточно разбираетесь в Git, такие не-атомарные изменения потом нужно объединить в один коммит с помощью [interactive rebase](https://git-scm.com/book/ru/v1/%D0%98%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D1%8B-Git-%D0%9F%D0%B5%D1%80%D0%B5%D0%B7%D0%B0%D0%BF%D0%B8%D1%81%D1%8C-%D0%B8%D1%81%D1%82%D0%BE%D1%80%D0%B8%D0%B8" \l "%D0%98%D0%B7%D0%BC%D0%B5%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5-%D1%81%D0%BE%D0%BE%D0%B1%D1%89%D0%B5%D0%BD%D0%B8%D0%B9-%D0%BD%D0%B5%D1%81%D0%BA%D0%BE%D0%BB%D1%8C%D0%BA%D0%B8%D1%85-%D0%BA%D0%BE%D0%BC%D0%BC%D0%B8%D1%82%D0%BE%D0%B2) и [squash](https://git-scm.com/book/ru/v1/%D0%98%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D1%8B-Git-%D0%9F%D0%B5%D1%80%D0%B5%D0%B7%D0%B0%D0%BF%D0%B8%D1%81%D1%8C-%D0%B8%D1%81%D1%82%D0%BE%D1%80%D0%B8%D0%B8" \l "%D0%A3%D0%BF%D0%BB%D0%BE%D1%82%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5-%D0%BA%D0%BE%D0%BC%D0%BC%D0%B8%D1%82%D0%BE%D0%B2).

Итак, после редактирования файлов мы имеем следующую ситуацию (это вывод git status):

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | On branch fix-protobaz  Changes not staged for commit:  (use "git add <file>..." to update what will be committed)  (use "git checkout -- <file>..." to discard changes in working directory)  modified: src/protobaz.rs  no changes added to commit (use "git add" and/or "git commit -a") |

В выводе есть все необходимые вам команды:

* git add <file>... добавляет файл в содержимое коммита, который вы собираетесь записать
* git checkout -- <file>... откатывает ваши изменения файла

Поэтому делаем git add src/protobaz.rs, а затем git commit. Откроется редактор, в котором нужно ввести сообщение коммита.

Сообщение коммита — это описание того, что вы сделали. Его читают другие участники проекта и рецензент. Поэтому оно должно быть осмысленным и читаемым.

Формат сообщения о коммите таков:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | Краткое описание коммита (не более 50 символов)  Подробное описание коммита - зачем он сделан, почему нельзя сделать по-другому,  источники информации.  Служебная информация - теги, ссылки на задачи, какие задачи коммит закрывает. |

Например,

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | Исправляем длину буфера в реализации PROTOBAZ  Неверное вычисление длины приводило к отсечению части сообщения в ситуации,  когда пользователь использует foobar-1.3.4.  Смотри http://foobar.org/bugs?id=1234  Fix #123 |

Из всех частей сообщения, только первая является обязательной.

Она должна иметь вид <Глагол в настоящем времени, первом лице, множественном числе> <объект изменения>. Говорим о том, что мы делаем в этом коммите: [мы] исправляем ошибку, [мы] добавляем возможность, [мы] обновляем документацию.

Последняя строка — это команда GitHub. Когда коммит с такой командой попадает в master, GitHub автоматически закроет указанную задачу. Можно использовать разные формы этой команды: Fix #123, fixes #123, close #123 и другие. Это экономит время на поддержку проекта.

В наших проектах нужно использовать Fix #123 или Close #123 на последней строке сообщения коммита.

git log --oneline выводит историю в формате «1 коммит — 1 строка на экране». При этом он использует в качестве описания коммита первую строку — краткое описание. Поэтому оно обязательно должно быть отделено пустой строкой от остального описания — иначе однострочный вывод разъедется.

Язык сообщения о коммите должен соответствовать принятому языку проекта. Поскольку наши проекты нацелены на русскоязычную аудиторию и разработчики говорят по-русски, сообщения коммитов также должны быть на русском.

Когда вы ввели сообщение коммита в редакторе, сохранили файл и закрыли его, можно выполнить git log и [убедиться](https://git-scm.com/book/ru/v1/%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-Git-%D0%9F%D1%80%D0%BE%D1%81%D0%BC%D0%BE%D1%82%D1%80-%D0%B8%D1%81%D1%82%D0%BE%D1%80%D0%B8%D0%B8-%D0%BA%D0%BE%D0%BC%D0%BC%D0%B8%D1%82%D0%BE%D0%B2), что коммит записан в историю.

Проверяем изменения

Когда вы сделали правки, стоит их проверить — если только это не что-то абсолютно тривиальное.

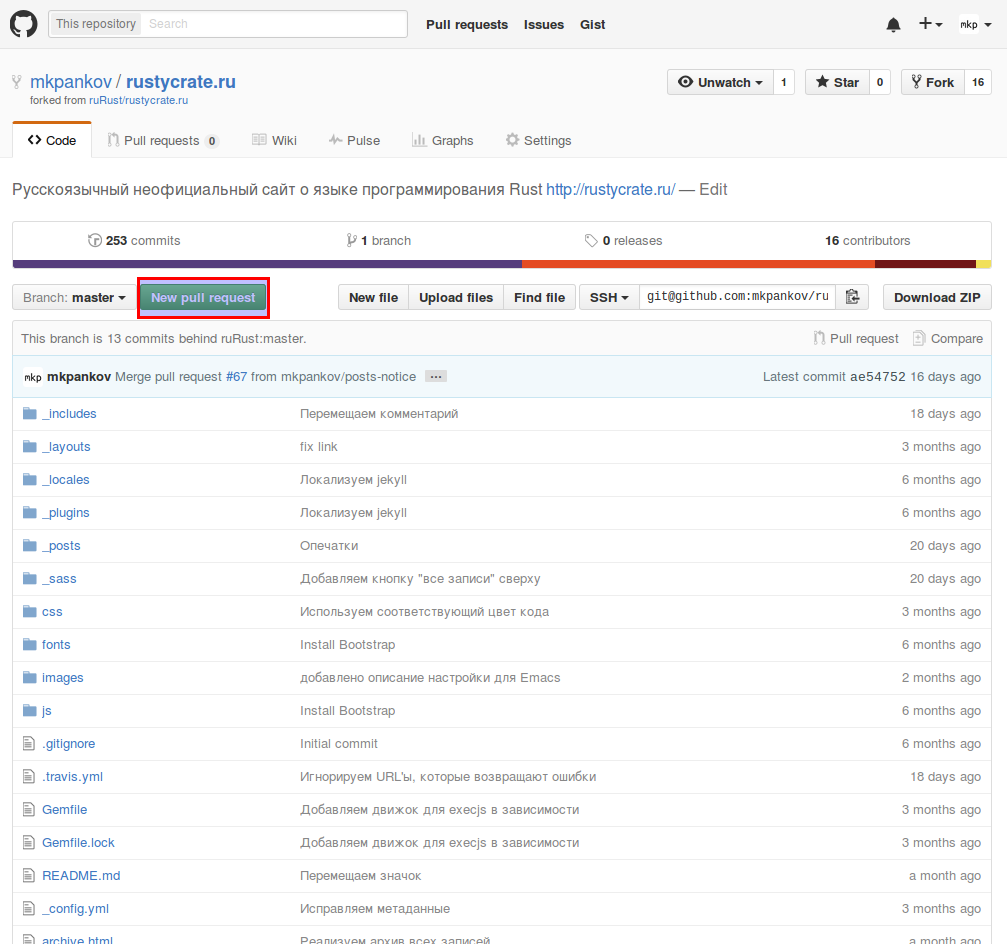
Для этого нужно собрать проект и запустить тесты, если они есть. В любом случае стоит проверить работу кода, который вы написали или изменили, запустив программу или вызвав библиотеку.

Если проект — это статически генерируемый сайт, то сгенерируйте его локально и убедитесь, что ничего не отвалилось и вёрстка не разъехалась. Если книга — то же самое. Смотрите по крайней мере на те места, которые вы правили.

Создаём Pull Request

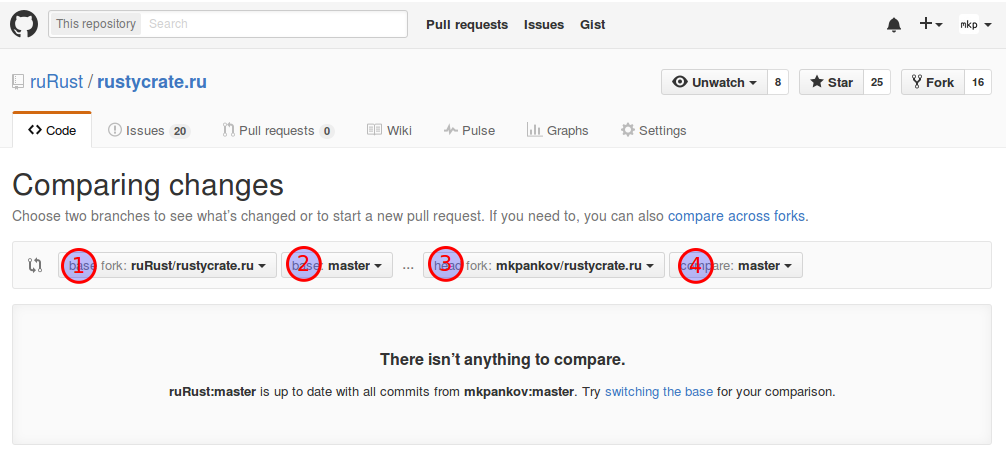
Когда работа и проверка закончены, пора создавать Pull Request. Pull Request — это запрос на вливание изменений из вашей ветки в основную ветку исходного репозитория. Таким образом они попадут к хозяевам проекта.

Чтобы создать Pull Request, зайдём на страницу вашего форка. Справа от выпадающего меню с выбором ветки есть кнопка «New pull request».



Нажимаем её.

Вы попадаете в окно сравнения веток.

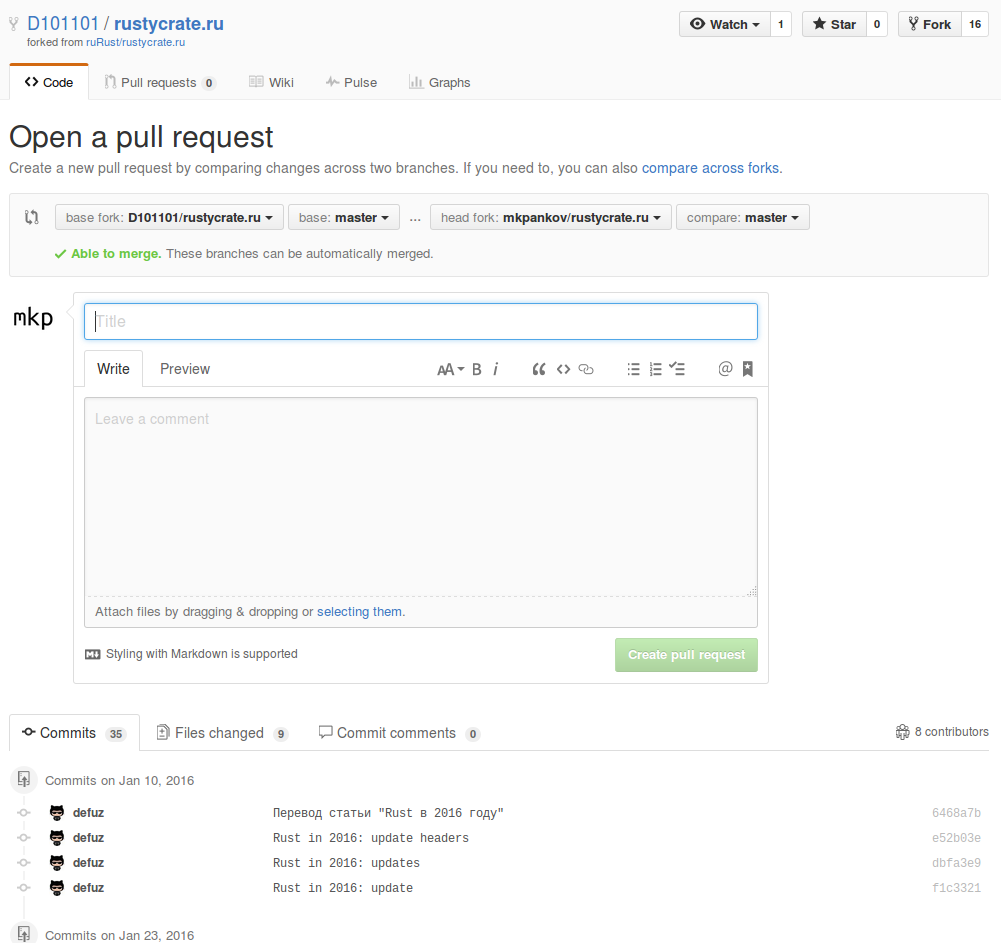


Вот элементы этого окна, по порядку:

1. Базовый репозиторий, в который будет создаваться PR. Это должен быть репозиторий, от которого вы делали форк. Если вы форкнули проект ruRust/rustycrate.ru, а ваше имя пользователя GitHub — user, то у вас будет проект user/rustycrate.ru.
2. Базовая ветка в этом репозитории, обычно master.
3. Репозиторий, откуда должны вливаться изменения. Здесь должен быть выбран репозиторий в вашем аккаунте — user/rustycrate.ru.
4. Ветка, откуда будут вливаться изменения. Это должна быть ветка, которую мы создали в разделе «Создаём ветку».

Дальше просмотрите изменения — то ли это, что вы делали? Если да, то нажимайте кнопку «Create pull request». В моём примере её нет, т. к. ветки в форке и в оригинале находятся в одинаковом состоянии. В вашем же случае внизу будет список коммитов, которые попадут в исходный репозиторий, и, на других вкладках — сами изменения и комментарии к изменениям.

После нажатия кнопки появится окно ввода сообщения Pull Request.



Сообщение PR — это описание того, что сделано и зачем. В отличие от сообщения коммита, здесь уже нужно писать высокоуровневое описание того, какие изменения сделаны. В части «зачем», а также по формату самого сообщения — стоит придерживаться тех же правил, что и в случае с коммитами. Короткий заголовок (Title), в Comment — описание, а затем служебная информация (Fix #100500). Если вы писали команды закрытия задач в коммитах, здесь можно их не дублировать. Если нет — напишите здесь.

Затем нажимаем «Create pull request». Он создаётся, о нём приходит уведомление людям, поддерживающим проект, и он становится виден в исходном репозитории на вкладке «Pull requests». С этого момента начинается рецензирование изменений (code review).

Подсказка: если сразу после того, как вы отправили ветку в свой репозиторий (git push origin) зайти на страницу репозитория, там будет предложение создать Pull Request на вливание недавно отправленной ветки в master. Сделать это можно как в вашем форке, так и в исходном репозитории. Это будет отдельная кнопка вверху, и при её нажатии в качестве ветки для слияния будет указана та, куда вы делали git push.

Участвуем в Code Review

На этом этапе люди, поддерживающие проект, смотрят на сделанный вами Pull Request и обсуждают его с вами. Возможно, какие-то вещи нужно будет исправить, улучшить, или доделать в процессе рецензирования — в этом нет ничего необычного, и это не значит что вы плохо поработали.

Со стороны автора Pull Request (а раз вы читаете это руководство, вы наверняка автор) требуется с пониманием относиться к комментариям рецензента — они направлены на повышение качества проекта. Если ошибка в сборке выявляется в процессе рецензирования, это гораздо лучше, чем если она попадёт в репозиторий, а следующий человек, который попытается поучаствовать в проекте, не сможет этого сделать из-за той самой ошибки. Да и это касается не только сборки, разумеется.

Также отмечу, что общаться в комментариях к PR следует вежливо и с уважением. Даже если вы общаетесь не с рецензентом. Не надо делать разнообразные далекоидущие выводы относительно пола, возраста, расы, интеллектуальных способностей и других черт человека. Если вы будете грубить рецензенту, в следующий раз он просто не захочет смотреть PR от вас. Не забывайте, что рецензирование — это тоже работа. Так что будьте добры и уважайте труд людей — они делают это бесплатно.

Если кто-то ведёт себя неадекватно — не медлите. Сначала сообщите об этом собеседнику и призовите его к благоразумию. Если не сработало — смело обращайтесь к рецензенту или к автору данного текста (Панкову Михаилу — @mkpankov). Это можно сделать в нашем [чате](https://gitter.im/ruRust/general).

Пожелание относительно процесса рецензирования — постарайтесь не сильно затягивать с ответами на комментарии или изменением указанных вещей.

Почему это важно? Автор PR хорошо разбирается в том, что он сделал, а если процесс затягивается на недели и месяцы, высока вероятность, что или автор, или рецензент отвлекутся на другие вещи, а обратный вход в контекст изменений тоже стоит усилий и времени. «Другие вещи» здесь — это вся остальная жизнь человека. Может на работе аврал, может личные проблемы, может заболел. В итоге может получиться так, что ваш PR навеки повиснет в неготовом состоянии и так и не будет влит. Конечно, нам бы этого не хотелось.

В отдельных случаях стоит обсудить спорный момент в чате, нежели перебрасываться комментариями на GitHub. Это гораздо быстрее, и объяснить свою позицию в интерактивном общении проще.

Также стоит отметить, что процесс рецензирования итеративен по природе. То, что рецензент написал комментарий к какой-то строке на пятой итерации, хотя она не менялась с первой, не значит, что он захотел придраться. Возможно, в результате окружающих изменений она потеряла смысл, или нужно изменить стиль фразы в документации, чтобы он согласовался с окружающим текстом. Также, часто просто сложно увидеть все места, требующие правки, с первого раза — внимательность и память любого человека ограничены. По мере того, как самые заметные вещи исправляют, внимание уделяется менее заметным — это нормально.

В нашем сообществе не принято наезжать на новичков, и мы всегда стараемся помочь. Не пугайтесь рецензирования — выше написано много слов, но на деле достаточно просто адекватно себя вести и не забивать на исправления.

Когда этот этап завершается, рецензент нажимает кнопку «Merge Pull Request». Ваши изменения влиты в основной репозиторий проекта.

Поздравляем! Теперь вы полноправный участник проекта.

Завершение работы

После вливания PR нужно прибраться в репозитории. Если ваши изменения самодостаточны и после PR не требуется продолжать работу дальше, стоит удалить ветку. Как вы помните, мы создали её раньше, чтобы удобнее управлять изменениями.

Но сначала лучше обновить вашу локальную master-ветку — тогда git убедится, что ваша ветка уже влита в master, и не будет предупреждать, что вы можете потерять свои изменения.

Итак, обновляем нашу локальную рабочую копию. Для этого добавим ещё один remote — так называется удалённый репозиторий. Сейчас он у вас только один — origin, и он указывает на ваш форк. Добавим remote с именем upstream, который будет указывать на исходный репозиторий:

|  |  |
| --- | --- |
| 1 | git remote add upstream https://github.com/ruRust/rustycrate.ru.git |

URL в конце — это URL того репозитория, который вы форкнули. (На всякий случай — он написан в небольшом поле справа от переключателя «HTTPS-SSH», правее зелёной кнопки «New pull request».)

Добавив его, можно обновить наш локальный master:

|  |  |
| --- | --- |
| 1  2 | git checkout master  git pull --rebase upstream/master |

А теперь можно удалить нашу ветку fix-protobaz локально:

|  |  |
| --- | --- |
| 1 | git branch -d fix-protobaz |

и на сервере:

|  |  |
| --- | --- |
| 1 | git push --delete origin fix-protobaz |

Теперь у нас чистый, обновлённый репозиторий. Можно начинать работу над новым PR.

Если у вас что-то не получилось сделать, или непонятно написано — пишите об этом в [чате](https://gitter.im/ruRust/general) или создавайте задачи в нашем [репозитории](https://github.com/ruRust/rustycrate.ru).

Если в какой-то момент возникают ошибки — сделайте git status. Очень часто git сам подсказывает, что можно сделать и как это сделать. Если всё равно не получилось — обращайтесь в чат, поможем.

После этой публикации остаются неотвеченные вопросы: как поступать, если случайно закоммитил не в ту ветку; что делать, если кажется, что git потерял ваши изменения; как работать с сервисом code review Reviewable; как рецензировать чужие изменения и, возможно, другие.

ДЗ

1. PR к тимуру. приложить инструкции по работе с гитом

2.приложить файл md со своими стремлениями

3. сделать там блок схему. нужен двухмерный массив

array 0;0

j

git pull

git push

PR- предлагает свои изменеия

create fork

Code – копировать путь

в VS Code в терминале git clone путь

В проводнике нажать правой кнопкой на скачанной папке и нажать «открыть во встроенной терминале»

git checkout –b new\_branch

! не использовать ветку main в чужом репозитории!!

внести изменения в ветке

git commit –am “commit”

git push

git push –set ……………….(copy)

перейти в браузер, обновить свою страницу в GH

выбрать ветку new\_branch

нажать pull requests – зеленую new pull requests

сменить в compare ветку с main на new\_branch

зеленая create pull request. можно добавить комментарии