

Практическая работа №1

Основы Git и Github

Цель работы.

Выполнение практической работы направлено на изучение:

1. наиболее распространенных практик в области контроля версий программного обеспечения, его использования в командной разработке ПО и DevOps;
2. концепции Git, основанной на понятиях репозитория и ветвления версий ПО;
3. порядка использования GitHub и его базовых операций.

Основные идеи и теоретические основы.

Разработчики приложений редко работают в одиночку. В крупных проектах в области веб-/облачных/мобильных приложений, в проектах искусственного интеллекта, обычно задействовано множество людей – разработчики интерфейса, серверной части, баз данных и многие другие. Вносимые каждым участником изменения обязательно должны отслеживаться и контролироваться руководителем, чтобы исключить значимые ошибки в условиях совместной работы. Для организации такой работы применяются технологии распределенного контроля версий.

История. Разработка Linux в начале 2000-х годов осуществлялась в рамках бесплатной системы, известной как BitKeeper. В 2005 году BitKeeper превратился в платную систему, что создало определенные проблемы для разработчиков Linux. Линус Торвалдс возглавил команду по разработке новой системы контроля версий исходного кода. Проект реализовывался в короткие сроки, а его ключевые характеристики были определены в следующем виде:

- поддержка нелинейного развития (в то время патчи для Linux приходили со скоростью 6,7 патчей в секунду, необходимо было обеспечить эффективное решение связанных с данной особенностью задач);
- распределенная разработка: каждый разработчик должен иметь локальную копию полной истории разработки;
- совместимость с существующими системами и протоколами, что необходимо для обеспечения разнообразия сообщества Linux;
- эффективное управление крупными проектами;
- аутентификация истории, что гарантирует, что все распределенные системы будут иметь идентичные обновления кода;
- подключаемые стратегии слияния: множество путей развития могут привести к сложным интеграционным решениям, которые могут потребовать четких интеграционных стратегий.

Распределенные системы контроля версий (Distributed Version Control Systems, DVCS) стали важнейшими инструментами разработки программного обеспечения для совместной разработки ПО. Их используют не только инженеры-программисты и специалисты DevOps, но и многие другие специалисты, в том числе, в области Data Science. Они полезны везде, где требуется отслеживание изменений/версий и/или сотрудничество между несколькими пользователями.

Несмотря на то, что существует множество распределенных систем управления версиями, Git является одной из самых популярных, а GitHub – наиболее популярной платформой контроля версий на базе Git. Популярность Git и GitHub делает их важным навыком для специалистов, связанных с разработкой ПО.

В данной практической работе ставятся следующие задачи:

1. развитие концептуальных и практических навыков для работы с Git и GitHub, начиная с обзора Git и GitHub, создания учетной записи GitHub, репозитория проекта, добавления в него файлов, работы с помощью веб-интерфейса;
2. знакомство с базовыми процессами Git, включающими `branche`, `pull requests` и `merge`, операции `fork` и `clone` для общедоступных репозиториях, `pull` и `push` для синхронизации базы кода между локальными и удаленными репозиториями;
3. практика работы с командами Git для процесса совместной разработки.

Система контроля версий позволяет отслеживать изменения в ваших документах. Это позволяет восстановить старые версии документа, если была допущена ошибка и значительно упрощает сотрудничество с другими разработчиками. Git – бесплатное программное обеспечение с открытым исходным кодом, распространяемое по лицензии GNU General Public License. Git – это распределенная система контроля версий, что означает, что пользователи в любой точке мира могут получить копию вашего проекта на своем компьютере. После того, как они внесли изменения, они могут синхронизировать свою версию с удаленным сервером, чтобы поделиться ею с вами. Git – не единственная существующая система контроля версий, но данная технология фактически единственная, которая обеспечивает аспект распределенной работы, что и стало основной причиной ее значительной популярности. Системы контроля версий широко используются для задач, связанных с кодом, но вы также можете управлять изображениями, документами и любыми другими типами файлов. Вы можете использовать Git без веб-интерфейса, используя интерфейс командной строки.

GitHub – один из самых популярных веб-сервисов для репозиториях Git. К популярным также необходимо отнести GitLab, BitBucket и Beanstalk. Что особенного в модели Git Repository?

- Git спроектирован как распределенная система контроля версий.
- Git в первую очередь сосредоточен на отслеживании исходного кода во время разработки.
- Git содержит элементы для координации действий программистов, отслеживания изменений и поддержки нелинейных рабочих процессов.

Git – это распределенная система контроля версий, которая используется для отслеживания изменений контента. Она служит центральной точкой для сотрудничества. В централизованной системе контроля версий каждый разработчик должен извлечь код из центральной системы и снова внести в него изменения. Поскольку Git представляет собой распределенную систему контроля версий, у каждого разработчика есть локальная копия полной истории разработки, и изменения копируются из одного такого репозитория в другой. Каждый разработчик может выступать в роли хаба (центрального узла).

При правильном использовании Git существует основная ветка (`branch`), соответствующая коду, запускаемому по умолчанию. Команды могут постоянно интегрировать изменения, готовые к выпуску, и одновременно работать над отдельными ветвями между выпусками в свет основной. Git также позволяет централизованно администрировать задачи с контролем уровня доступа для каждой команды. Git может сосуществовать локально, например, через клиент GitHub Desktop, или его можно использовать напрямую через браузер, подключенный к веб-интерфейсу GitHub.

GitHub – это онлайн-хостинг для репозиториях Git. GitHub – продукт дочерней компании Microsoft. GitHub предлагает бесплатные, профессиональные и корпоративные учетные записи. По состоянию на август 2019 года у GitHub было более 100 миллионов репозиториях.

Репозиторий – это структура данных для хранения документов, включая исходный код приложения. Репозиторий может отслеживать и поддерживать контроль версий.

Стоит остановиться также на GitLab. GitLab – это полноценная платформа DevOps, представленная в виде одного приложения. GitLab предоставляет доступ к репозиториям Git, контролируемый системой управления исходным кодом. С GitLab разработчики могут:

- Просматривать код, оставляя комментарии и помогая улучшать код друг друга.
- Работать со своей локальной копией кода.
- Осуществлять ветвления и слияния при необходимости.
- Оптимизировать тестирование и доставку с помощью встроенной непрерывной интеграции (CI) и непрерывной доставки (CD).

Прежде чем приступить к работе, вам необходимо знать несколько основных терминов:

Протокол SSH – метод безопасного удаленного входа с одного компьютера на другой.

Репозиторий в простом варианте изложения является местом, которое содержит папки вашего проекта, для которых настроен контроль версий.

Форк / fork – копия репозитория.

pull request – способ запроса на то, чтобы кто-то рассмотрел и утвердил предложенные вами изменения.

working directory содержит файлы и подкаталоги на вашем компьютере, которые связаны с Git репозиторием.

Есть несколько основных команд Git, которые вы будете использовать. Когда вы начинаете работу с новым репозиторием, его создание происходит только один раз: либо локально с последующей отправкой на GitHub, либо путем клонирования существующего репозитория с помощью команды «git init».

git add перемещает изменения из рабочего каталога в промежуточную область.

git status позволяет вам видеть состояние вашего рабочего каталога и промежуточный snapshot ваших изменений.

git commit делает снимок / snapshot изменений и фиксирует их в проекте.

git reset отменяет изменения, внесенные вами в файлы в вашем рабочем каталоге.

git log позволяет просматривать предыдущие изменения в проекте.

git branch позволяет вам создать изолированную среду в вашем репозитории для внесения изменений.

git checkout позволяет просматривать и изменять существующие изолированные среды (*branch'и*).

git merge позволяет соединить *branch'и* воедино.

Все файлы в GitHub хранятся в отдельных *branch*. Разработчики работают над исходными файлами в *branch*. *Master branch* является основной. Она хранит развертываемую версию вашего кода. *Master branch* создается по умолчанию, однако, вы можете использовать любую ветку в качестве основной, готовой к запуску кода. Когда вы планируете что-то изменить, вы создаете новую *branch* и даете ей понятное имя. Новая *branch* создается как точная копия исходной *branch*. Когда вы вносите изменения, созданная вами *branch* сохраняет изменения.

Чтобы изменить содержимое файла в теории Git необходимо не просто внести изменения, но и выполнить *commit* изменений. Когда разработчик завершает порученную ему работу, для сохранения изменений он выполняется т.н. *commit* / фиксацию кода. *Commit* означает, что разработчик убежден, что код представляет собой стабильную основу для разрабатываемой

функции или набора функций. Когда разработчик фиксирует измененный исходный код, он должен написать комментарий, описывающий изменения. Комментарий должен быть содержательным. Некоторые дополнительные «правила приличия»:

- Не заканчивайте сообщение при коммите точкой.
- Сообщения о коммитах должны содержать не более 50 символов – для предоставления подробной информации используйте расширенное окно.

Разработчики могут изменять исходные файлы в ветке, но изменения не будут опубликованы до утверждения. Порядок работы с утверждением:

- Подается команда `pull`.
- Код рассматривается и утверждается.
- Утвержденный код объединяется с основным кодом.

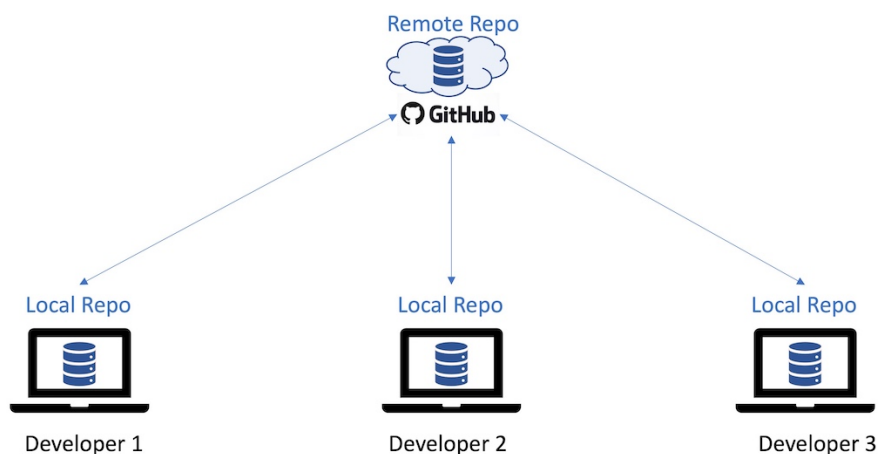
Когда все изменения в ветке завершены, эта ветка считается устаревшей и ее следует удалить.

Pull используется для инициирования слияния ветвей с целью фиксации изменений. *pull request* делает предложенные изменения доступными для просмотра и использования другими. Pull может следовать за любыми коммитами, даже если код еще не завершен. Для pull требуется, чтобы специальный пользователь утвердил изменения. Это может быть автор изменения или он может быть назначен из состава команды. Обратите внимание, что GitHub автоматически отправляет pull request от вашего имени, если вы вносите изменения в branch, которой вы не владеете. Поскольку файлы журналов неизменяемы, всегда можно найти человека, утвердившего внесение изменений. Чтобы открыть новый pull request в веб-интерфейсе GitHub необходимо выполнить следующую последовательность действий:

- Нажмите «Pull request» и выберите «New pull request».
- Выберите новую branch в поле сравнения.
- Прокрутите вниз, чтобы просмотреть изменения.
- Подтвердите, что изменения – это то, что вы хотите предоставить к утверждению.
- Добавьте заголовок и описание к запросу.
- Нажмите «Create pull request».

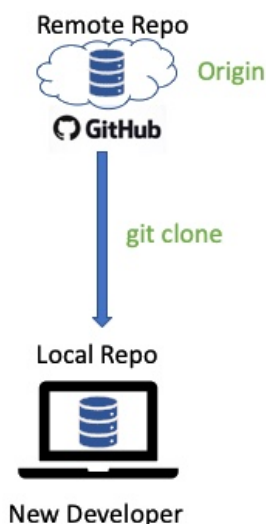
Forking и Cloning GitHub репозитория.

Системы контроля версий с распределенным исходным кодом, такие как GitHub, позволяют нескольким разработчикам проектов параллельно работать над его кодовой базой. Проект может находиться на GitHub.com в качестве общедоступного или частного репозитория. Каждый разработчик, работающий над проектом, может иметь на своих компьютерах собственные копии репозитория. Копия репозитория на компьютере разработчика является локальной для этого разработчика, и, следовательно, этот разработчик также называет это репозиторий своим *local repo*. Копия репозитория на GitHub.com находится на удаленном сервере, и, следовательно, для каждого разработчика это *remote repo*.

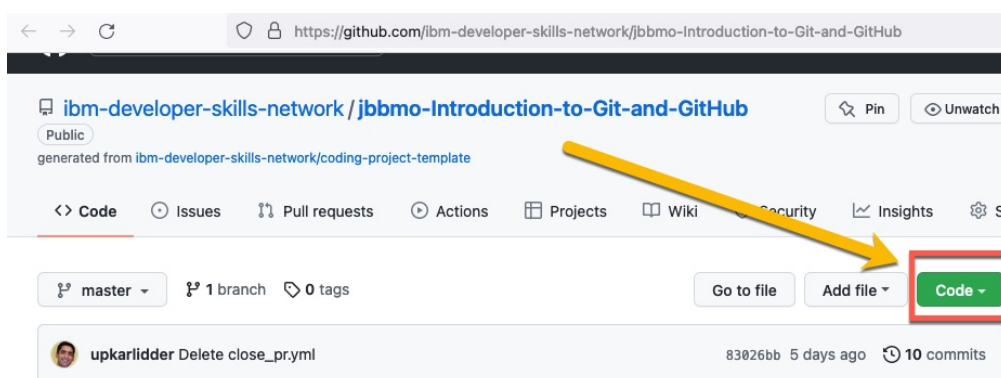


Clone

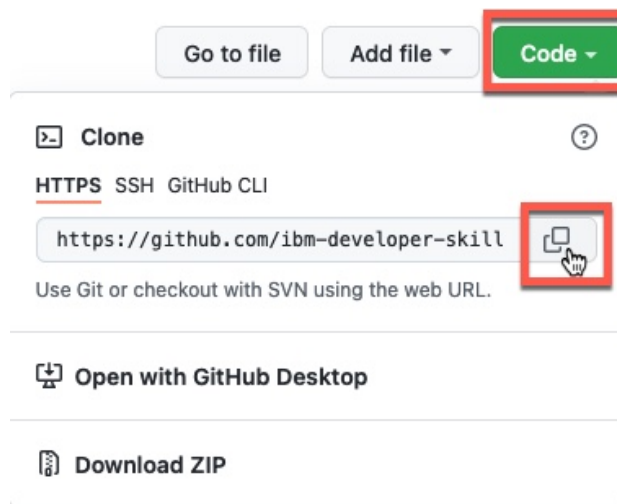
Допустим, к команде присоединяется новый разработчик для совместной работы над проектом. Этот разработчик может создать идентичную копию remote repo, используя операцию `git clone`. Удаленный репозиторий, из которого изначально клонирован проект, также называется `origin`.



Любой репозиторий в GitHub можно клонировать, перейдя в репозиторий и нажав кнопку «Code».



После этого у вас будет возможность получить всю кодовую базу удаленного репозитория несколькими способами, включая возможность скопировать URL-адрес HTTPS, а затем указать скопированный URL-адрес для выполнения команды `git clone URL` с вашего локального компьютера.



Создание ветвей и синхронизация изменений

После клонирования репозитория на локальный компьютер разработчик может начать вносить изменения в базу кода. Это может быть для таких задач, как добавление функций и улучшений или исправление ошибок. Обычно разработчик использует `git branch` для создания ветки, например, `feature1-branch`, делает эту ветку активной с помощью `git checkout` и вносит изменения в эту ветку, например, добавив или отредактировав файлы. Разработчик часто сохраняет свои изменения внутри ветки, используя `git add`, за которым следует `git commit`.

После того, как изменения для конкретной ветки завершены, вместо непосредственного слияния с основной веткой часто рекомендуется выполнить `push` новой ветки с изменениями в `origin`, где другие разработчики/рецензенты смогут протестировать/проверить изменения перед слиянием ветки с основной.

ПРИМЕЧАНИЕ. Поскольку в этом сценарии `feature1-branch` была разработана новым разработчиком проекта, у этого разработчика может не быть доступа для объединения своей ветки с основной веткой. Фактически, во многих проектах только сопровождающим или администраторам проекта разрешено объединяться с основной веткой, а в некоторых случаях может потребоваться экспертная оценка. Чтобы запросить проверку и объединение ваших изменений с основной веткой, многие проекты требуют отправки Pull Request (PR). Тогда как в некоторых случаях, например, если вы единственный разработчик проекта, этот шаг PR можно пропустить, и вы можете объединить и отправить свои изменения напрямую, если у вас есть доступ на запись к исходному репозиторию.

Время от времени разработчик может захотеть получить последнюю копию репозитория из `origin`, чтобы использовать ее в качестве основы для внесения изменений или проверки изменений другими. Например, это может произойти после того, как изменения в `feature1-branch` были отправлены в источник, и самостоятельный разработчик хочет просмотреть код. Для этой цели можно использовать команду `git fetch`.

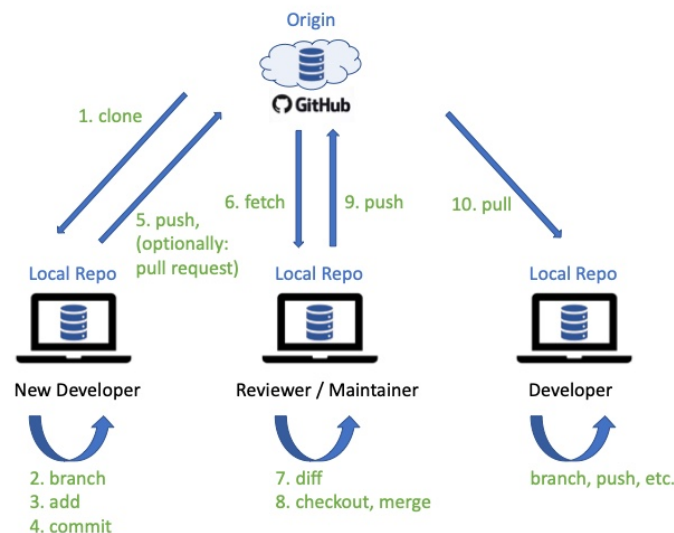
Команда `git diff` может помочь другим разработчикам, просматривающим ваш код, выявить и сравнить изменения. Как только рецензент или сопровождающий проекта просмотрит изменения и будет удовлетворен, он выполнит `git checkout` – перейдет в основную ветку, а затем `git merge` новую `feature1-branch`, которую затем можно будет удалить. После локального слияния ветки рецензент может с помощью `git push` обновленную основную ветку обратно в `origin`.

ПРИМЕЧАНИЕ. Команду «`git-remote -v`» можно использовать для проверки того, с какими удаленными репозиториями вы синхронизируете изменения `push` и `fetch`.

Другой вариант получения последней копии репозитория – использовать команду `git pull`. Команда `pull` по сути представляет собой комбинацию `fetch` и `merge`. То есть, используя эту единственную команду, вы можете как получить, так и объединить изменения в локальном репозитории.

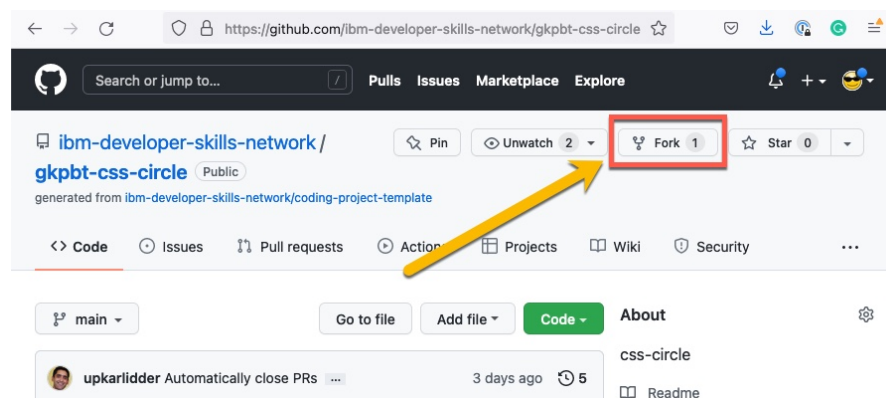
Например, другой разработчик, который хочет использовать обновленную кодовую базу с изменениями `feature1`, которые были объединены с основной веткой в исходном коде, может использовать команду `git pull`, чтобы `fetch` обновленную кодовую базу из исходного кода и `merge` его/ее локальную кодовую базу перед началом разработки.

Описанный здесь рабочий процесс клонирования->ветвления->слияния можно резюмировать на следующей диаграмме.



Fork

Если разработчик хочет создать производный проект с другим проектом в качестве отправной точки или работать над проектом, используя отдельный или независимый клон, он может выбрать `fork` проекта. Вы можете создать форк любого общедоступного проекта, перейдя на его страницу проекта на GitHub и нажав кнопку «Fork» в верхней части страницы.

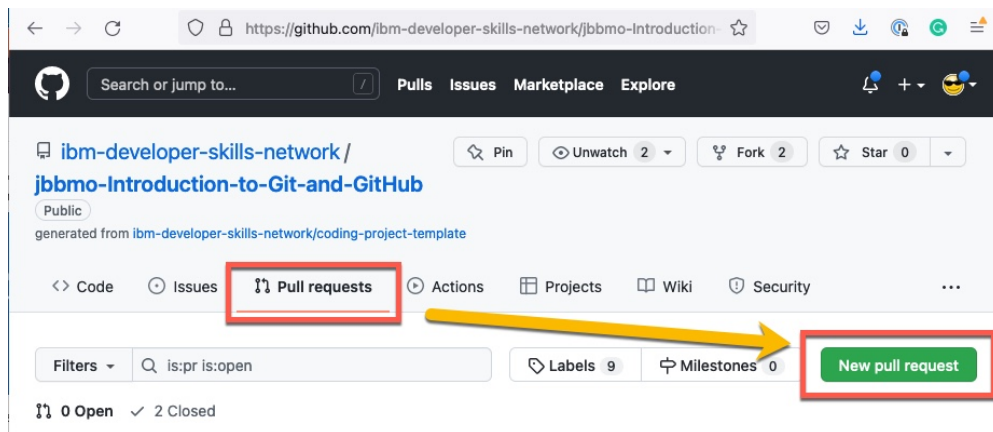


ПРИМЕЧАНИЕ. Параметр `fork` доступен только с использованием веб-интерфейса, и для создания разветвления не существует команды `git`.

Проект, из которого вы создаете ответвление (`fork`), называется `upstream` проектом.

После разветвления проекта разработчики, имеющие доступ к разветвлению, могут работать над обновлением и внесением изменений в разветвление, используя тот же рабочий процесс, который описан ранее, т.е. разветвленная копия проекта теперь становится источником, а разработчики, имеющие доступ к источнику, могут создавать его клоны на своих локальных машинах, где они могут создавать и объединять ветки, а также синхронизировать изменения с источником, используя `pull` и `push`.

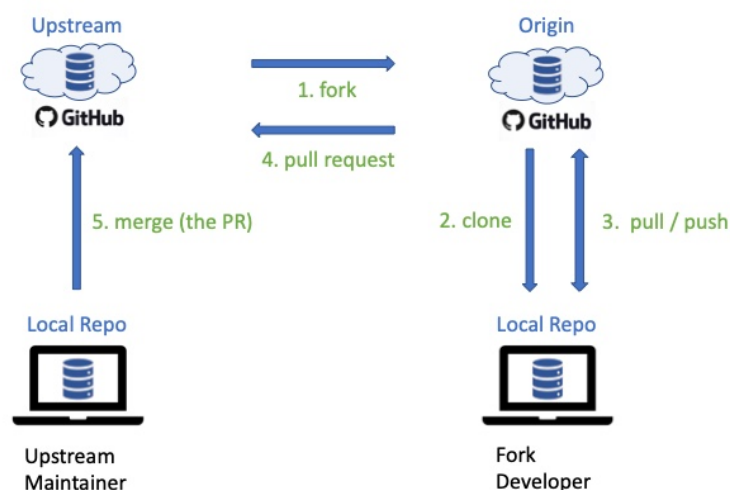
Однако важно отметить, что синхронизация изменений с использованием слияния и отправки может выполняться только с репозиториями, к которым у разработчиков есть доступ на запись, то есть в данном случае это их ответвление проекта, то есть *origin*, из которого они создают свои локальные клоны. Но что, если разработчик хочет внести свои изменения обратно в *upstream*, к которому у него нет доступа на запись? В этом случае они могут отправить *pull request* с предлагаемыми изменениями. Новый *pull request* можно создать, перейдя на домашнюю страницу проекта, перейдя на вкладку «Pull Requests» и нажав «New Pull Request».



ПРИМЕЧАНИЕ. Термин «*Pull Request*» не следует путать с командой `git pull`, которую вы используете для *fetch* и *merge* последней базы кода с вашим локальным репозиторием. *Pull Request*, как следует из названия, — это просто запрос на просмотр и *pull* предложенных вами изменений. В рамках PR вы предоставляете подробную информацию о предлагаемых изменениях и вашей реализации.

Сопровождающие вышестоящего (*upstream*) проекта могут просмотреть изменения в PR и решить, объединять их или нет. В некоторых случаях они могут предоставить отзыв (путем комментариев в PR) или попросить отправителя PR выполнить какое-либо разрешение конфликта, например, применив свои изменения к последней кодовой базе и повторно отправив PR.

Описанный здесь рабочий процесс `fork->clone->pr` кратко показан на рисунке ниже.



Когда форкать и клонировать?

Обычно, если у вас есть доступ к репозиторию проекта, например, как члену команды, совместно разрабатывающей кодовую базу, вы можете клонировать репозиторий и синхронизировать изменения из локальной копии репозитория, используя `pull` и `push`.

Однако, если есть общедоступный проект, в который вы хотите внести свой вклад, но у вас нет доступа для записи, или если вы используете общедоступный проект в качестве отправной точки для своего собственного проекта, вы можете разветвить проект. Затем поработайте с разветвленной кодовой базой, клонировав ее на свой компьютер и сотрудничая с командой разработчиков, работающей над разветвлением, используя синхронизацию pull-push с вашим разветвлением проекта. Но если вы хотите внести свои изменения обратно в вышестоящий проект (исходный проект, из которого вы создали ответвление), вы можете отправить свои изменения с помощью pull request – запроса на включение.

Порядок работы

Тренировочные задания.

1. Регистрация в GitHub.

Зайдите на сайт GitHub, <https://github.com> - <https://github.com/join>

Join GitHub

Create your account

Username *

Email address *

Password *

Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. [Learn more.](#)

Email preferences

☒ Send me occasional product updates, announcements, and offers.

Verify your account

Please solve this puzzle so we know you are a real person

Verify

Create account

By creating an account, you agree to the [Terms of Service](#). For more information about GitHub's privacy practices, see the [GitHub Privacy Statement](#). We'll occasionally send you account-related emails.

Определитесь с именем пользователя, введите адрес электронной почты, придумайте пароль, нажмите кнопку «Sign up for GitHub».

Далее необходимо пройти небольшой тест, чтобы доказать, что вы человек – нажмите «Verify» и решите представленную головоломку.

Verify your account

Please solve this puzzle so we know you are a real person

Verify

Затем нажмите «join a free plan», после чего вы попадете на экран, где сможете выбрать тип учетной записи — выберите бесплатную личную учетную запись (предлагается по умолчанию).

Email preferences

☒ Send me occasional product updates, announcements, and offers.

Join a free plan

GitHub задает несколько вопросов о вашей работе, опыте программирования и интересах.

What do you plan to use GitHub for?

(Select up to 3)



Learn to code



Learn Git and GitHub



Host a project (repository)



Create a website with
GitHub Pages



Collaborating with my team



Find and contribute to
open source



School work and student
projects



Use the GitHub API



Other

I am interested in:

languages, frameworks, industries

We'll connect you with communities and projects that fit your interests.

For example: `zeplin` `elm` `apm`

Complete setup

После этого вам необходимо ответить на полученное электронное письмо, подтверждающее, что вы связались с GitHub из учетной записи, к которой вы имеете доступ.



Please verify your email address

Before you can contribute on GitHub, we need you to verify your email address.

An email containing verification instructions was sent to **Your email address**

[Resend verification email](#)

[Change your email settings](#)

Далее вы можете создать репозиторий или организацию или пройти курс «Introduction to GitHub». Организация — это совокупность учетных записей пользователей, владеющих репозиториями. У организаций есть один или несколько владельцев, обладающих правами администратора для организации.

Сердцем проекта на основе Git является репозиторий. Он содержит весь ваш код и связанные с ним артефакты, включая такие вещи, как:

- Файл README, описывающий цель проекта.
- Файл LICENSE, в которой необходимо указать способы использования вашего кода иными пользователями.

Вы можете сделать свой репозиторий частным (доступным только для людей с учетными записями, имеющими разрешение на его просмотр) или общедоступным (доступным для поиска и просмотра всеми).

Когда вы создадите свой репозиторий, вы заметите, что он имеет несколько вкладок и открывается на вкладке «Code». Code – место, где находятся все исходные файлы. Git изначально создавался как хранилище исходного кода, и теперь сюда попадают самые разные файлы.

Если вы выбрали автоматическое создание README и/или LICENSE, они также будут на данной вкладке.

Issues позволяет отслеживать открытые элементы в базе вашего проекта.

Pull Requests – это часть механизма взаимодействия с другими пользователями. Pull requests определяют изменения, которые зафиксированы и готовы к проверке перед объединением в основную ветку.

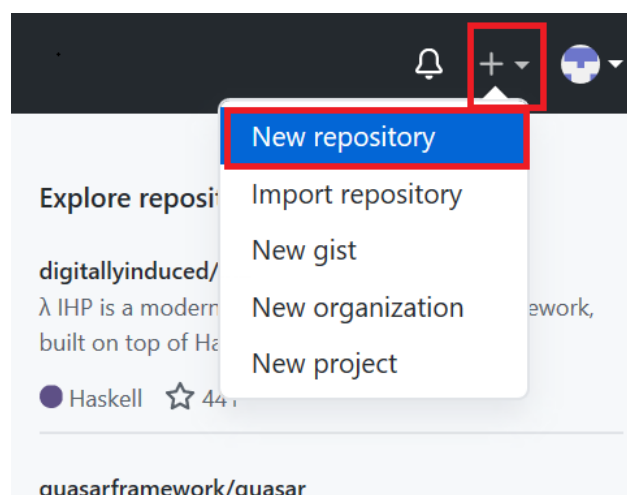
Projects – инструменты для управления, сортировки, планирования и т.д. ваших различных проектов.

Wiki, Security и Insights – эти инструменты часто оставляются для более опытных пользователей и обеспечивают основу для связи с внешним сообществом пользователей.

Settings – GitHub позволяет выполнять множество настроек, включая изменение имени вашего репозитория и контроль доступа.

2. Создание репозитория в GitHub

Нажмите «+», затем нажмите «New Repository».



Чтобы создать новый репозиторий, вам необходимо заполнить следующие данные:


- дать новому репозиторию имя;
- добавить описание репозитория;
- выбрать видимость репозитория – хотите ли вы, чтобы он был публичным или частным;
- выбрать опцию «Initialize this repository with a README».

Create a new repository


A repository contains all project files, including the revision history. Already have a project repository elsewhere?
[Import a repository.](#)

Owner *

Repository name *


 Malika-s

 /


testrepo 

Great repository names are short and memorable. Need inspiration? How about [urban-octo-waffle?](#)

Description (optional)

☒  **Public**

Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**


You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

☒ **Initialize this repository with a README**

This will let you immediately clone the repository to your computer.

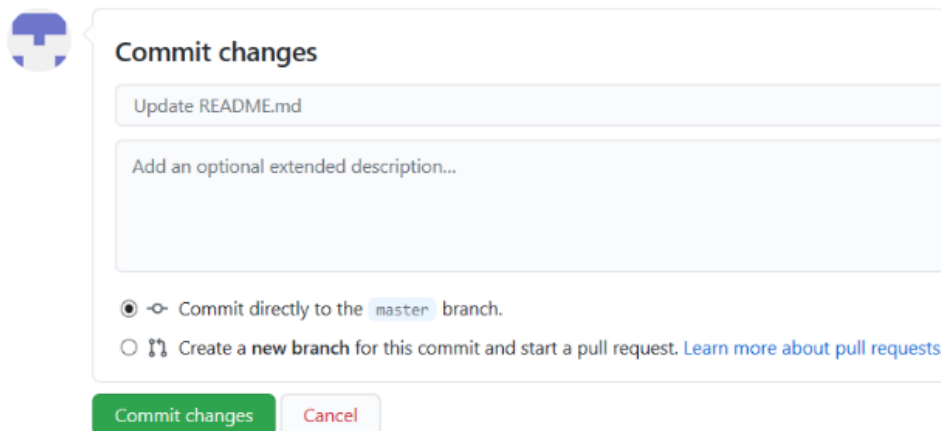
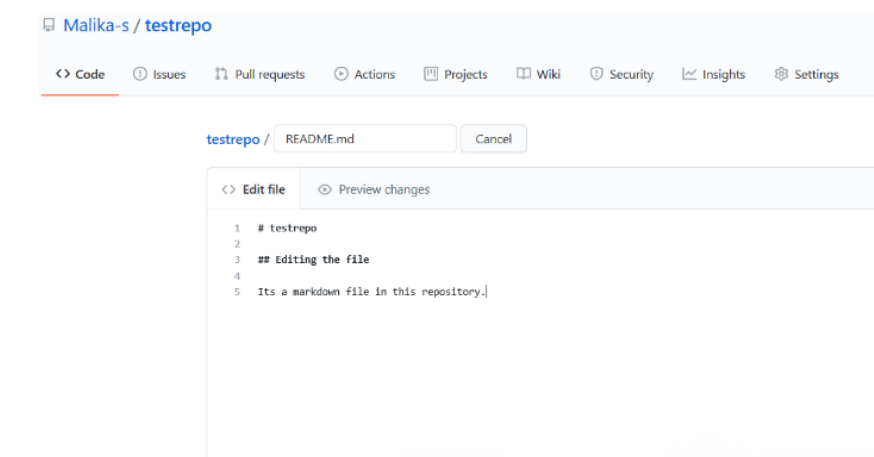
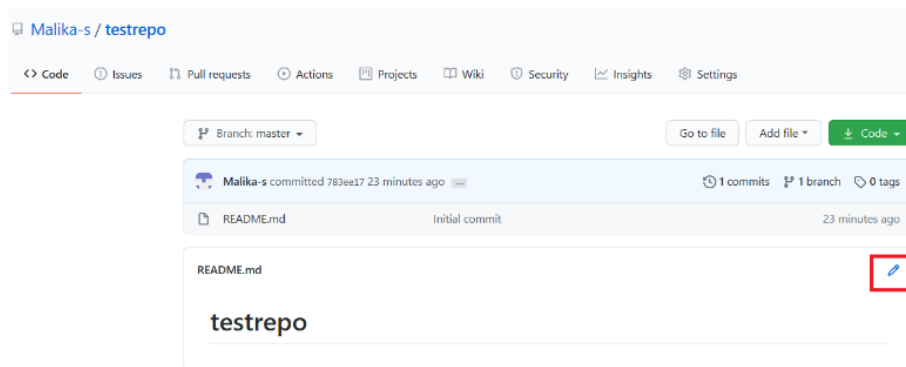
Add .gitignore: **None**

Add a license: **None** 

Create repository

Затем нажмите «Create Repository». Теперь вы будете перенаправлены в созданный вами репозиторий. Корневая папка вашего репозитория указана по умолчанию и содержит только один файл README.md.

Следующий шаг – отредактировать файл README. Это можно сделать в браузере: нажмите на карандаш, чтобы открыть онлайн-редактор, и вы сможете изменить текст файла README. Чтобы сохранить изменения в репозитории, вы должны зафиксировать их. После внесения изменений прокрутите вниз до раздела «Commit changes». Добавьте сообщение о фиксации и (при необходимости) описание, затем нажмите «Commit changes». «Commit changes» используется для сохранения изменений в репозитории. Вернитесь на главный экран, щелкнув ссылку на имя репозитория. Обратите внимание, что файл readme обновлен, и проверьте внесенные изменения.



Создание нового файла с помощью встроенного веб-редактора GitHub, который запускается в браузере. Нажмите «Add File», затем нажмите «Create New File», чтобы создать новый файл. Например, создаем файл Python с именем firstpython.py. Сначала укажите имя файла. Затем добавьте комментарий, описывающий ваш код, а затем добавьте код. После завершения зафиксируйте изменения в репозитории. Вы можете видеть, что ваш файл теперь добавлен в репозиторий, а в списке репозитория показано, когда файл был добавлен или изменен.

Malika-s / testrepo

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

Branch: master Go to file Add file Code

Malika-s committed 3861fb4 4 minutes ago 2 commits 1 branch 0 tags

README.md Update README.md 4 minutes ago

README.md

testrepo

Editing the file

Its a markdown file in this repository.

masterrepo.png (1730 x 888)

Malika-s / testrepo

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

Branch: master Go to file Add file Code

Malika-s committed 3861fb4 5 minutes ago 2 commits 1 branch 0 tags

README.md Update README.md 5 minutes ago

README.md

testrepo

Editing the file

Its a markdown file in this repository.

Create new file Upload files

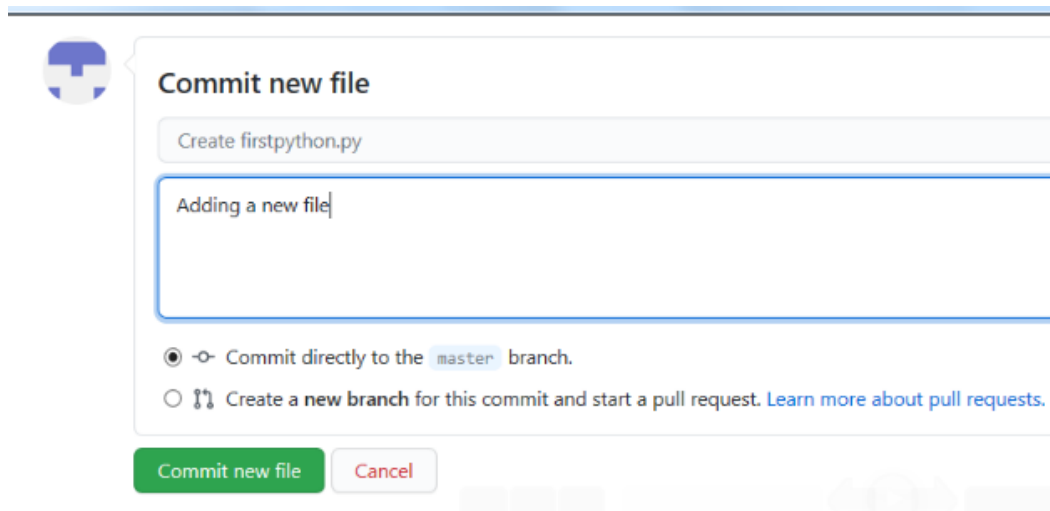
Malika-s / testrepo

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

testrepo / firstpython.py Cancel

<> Edit new file Preview

```
1 # Display the output
2 print("New Python file")
```



Commit new file

Create firstpython.py

Adding a new file

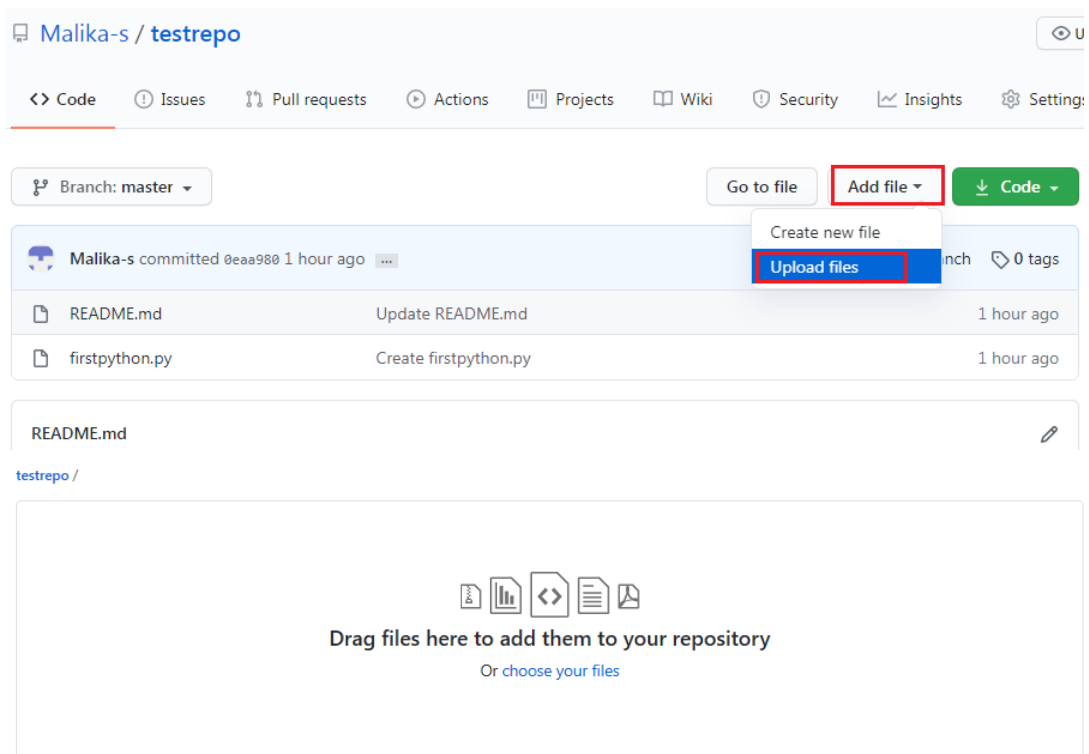
☒ Commit directly to the `master` branch.

☐ Create a new branch for this commit and start a pull request. [Learn more about pull requests.](#)

Commit new file Cancel

Если вам нужно изменить файл, необходимо выполнить следующее. Щелкните имя файла, а затем щелкните значок карандаша, внесите изменения и зафиксируйте их.

Вы также можете загрузить файл из вашей локальной системы в репозиторий. На главном экране репозитория нажмите «Add File» и выберите параметр «Upload files». Нажмите «Choose Your Files» и выберите файлы, которые вы хотите загрузить из своей локальной системы. Процесс загрузки файла может занять некоторое время, в зависимости от того, что вы загружаете. После завершения загрузки файлов нажмите «Commit Changes». В репозитории теперь отображаются загруженные файлы.



Malika-s / testrepo

<> Code ⓘ Issues 🔗 Pull requests ⚙️ Actions 📁 Projects 📖 Wiki 🛡️ Security 📊 Insights ⚙️ Settings

Branch: master

Go to file Add file Code

Create new file

Upload files

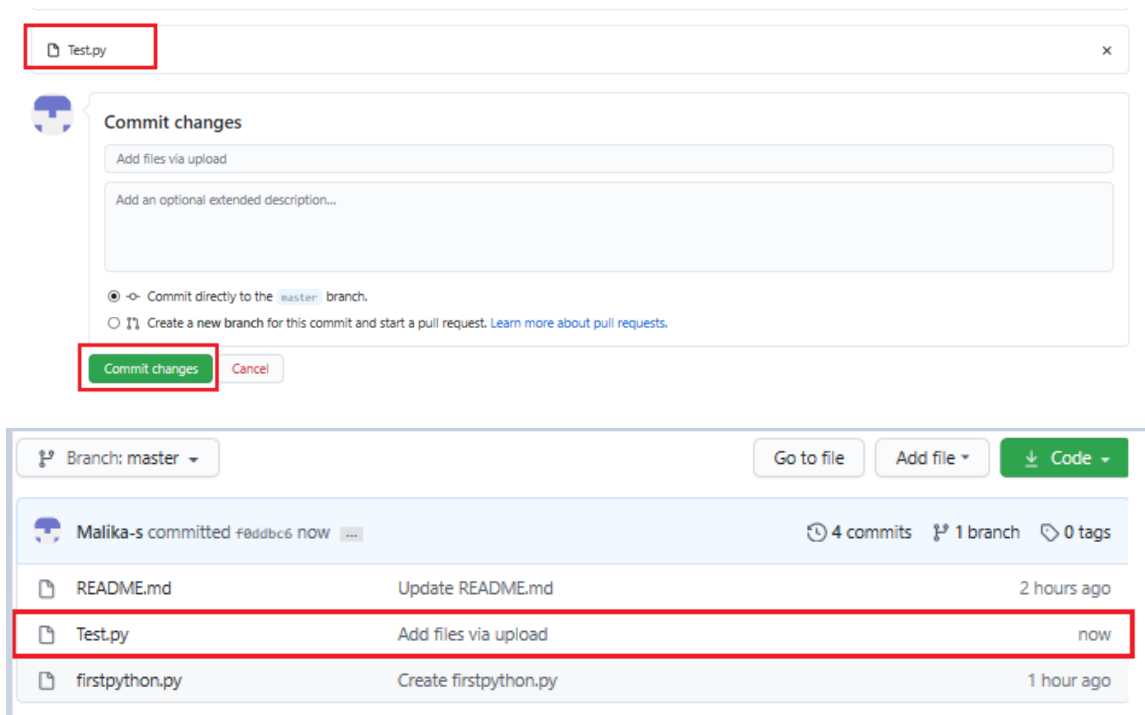
Malika-s committed 0eaa980 1 hour ago

README.md	Update README.md	1 hour ago
firstpython.py	Create firstpython.py	1 hour ago

testrepo /

Drag files here to add them to your repository

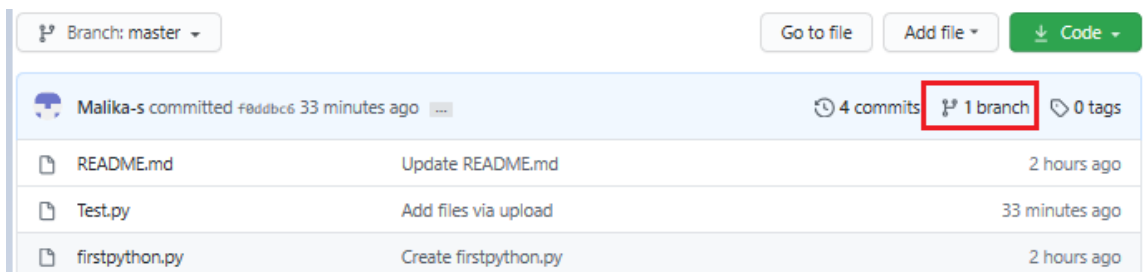
Or choose your files



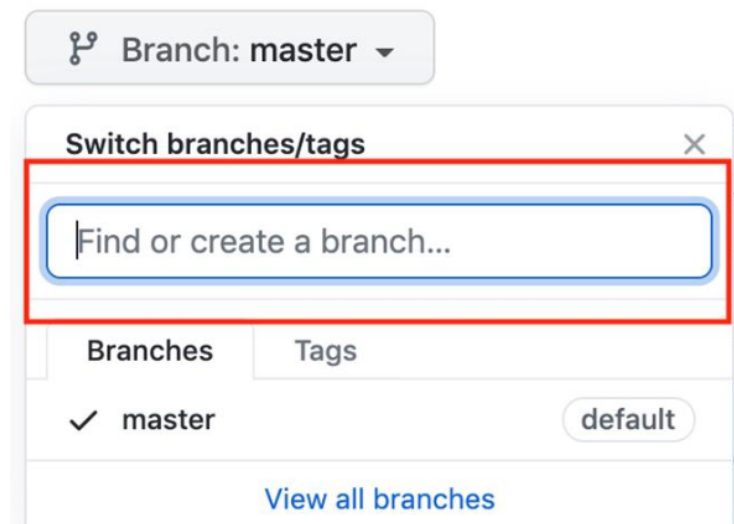
3. Создание branch

Чтобы добавить ветку в ваш репозиторий, выполните следующие шаги.

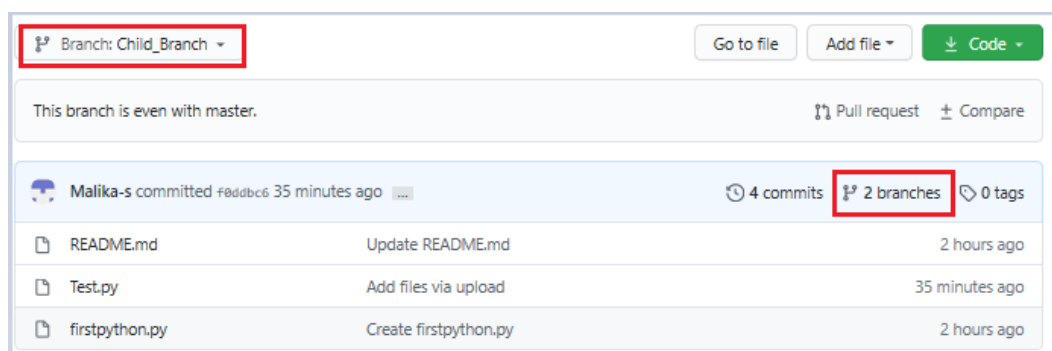
- Перейдите на главную страницу вашего репозитория. Обратите внимание, что когда вы создавали свой репозиторий, для вас была создана основная ветка.



- В верхней части списка файлов найдите раскрывающееся меню «branch». (По умолчанию в меню отображается «Branch: master».) Щелкните раскрывающееся меню, введите имя ветки, которую вы хотите создать, и нажмите Enter на клавиатуре.



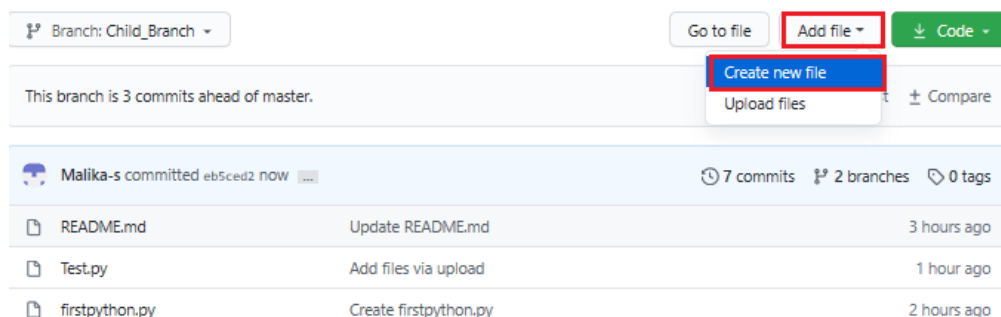
В вашем репозитории теперь есть две ветки: master и Child_Branch. Вы можете щелкнуть раскрывающееся меню, чтобы увидеть свои ветки.



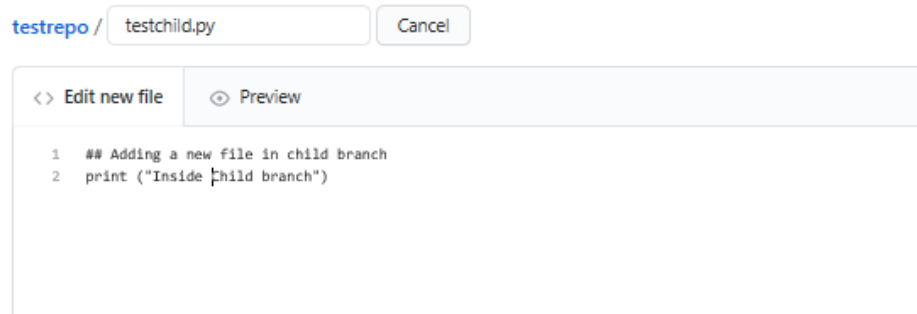
Все файлы, которые находились в основной ветке, теперь скопированы в Child_Branch. Обратите внимание: когда вы добавляете или редактируете файл в Child_Branch, это изменение не будет автоматически внесено в основную ветку.

Чтобы добавить файл в новую ветку, убедитесь, что Child_Branch (или любое другое имя, которое вы дали своей ветке) отображается в раскрывающемся меню «Branch», и выполните следующие шаги:

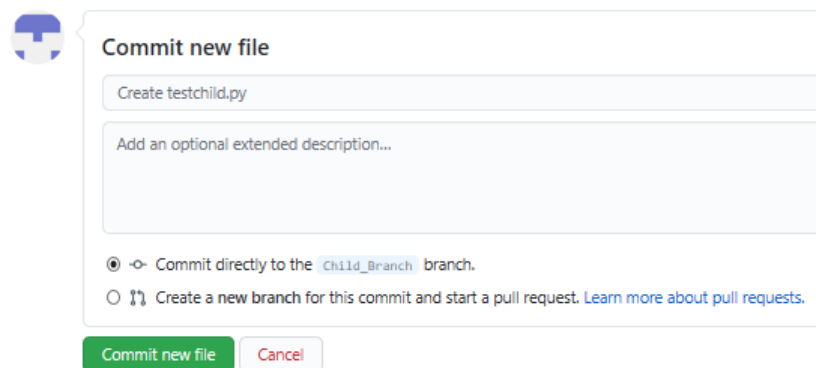
- Нажмите *Add file* -> *Create new file*, чтобы создать файл в репозитории.



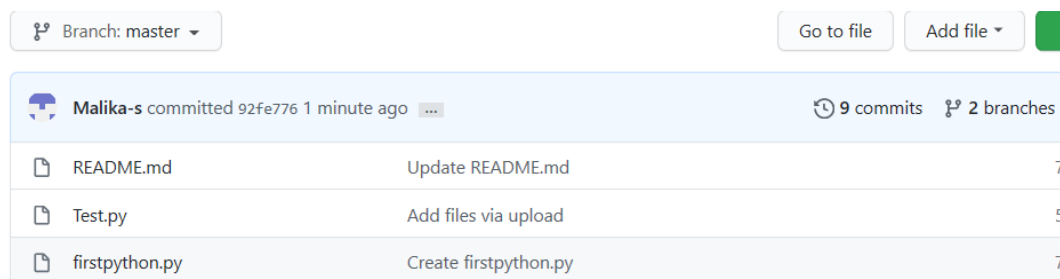
- Введите имя и расширение файла — например, testchild.py — и добавьте следующие строки в тело нового файла:



Прокрутите страницу вниз, добавьте описание файла, который вы собираетесь добавить (обратите внимание, что описание не является обязательным), и нажмите «Commit new file».

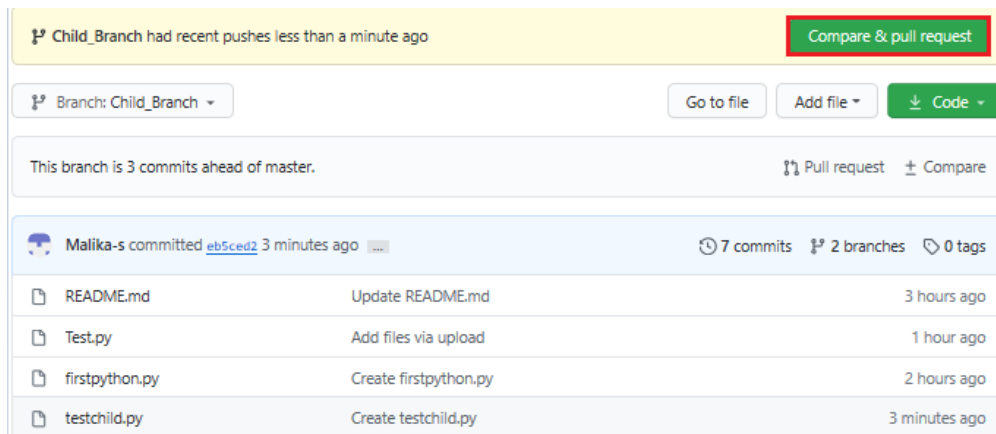


Файл, который вы добавили в дочернюю ветку, не добавляется автоматически в основную ветку. (Вы можете проверить это, используя раскрывающееся меню «Branch», чтобы перейти к основной ветке; обратите внимание, что в списке файлов нет файла testchild.py)

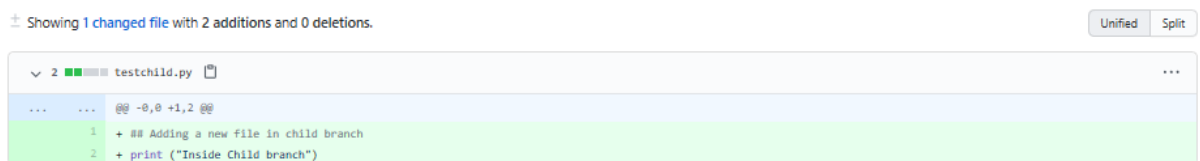


Вы также можете сравнить две ветки и открыть запрос на включение, который позволит вам скопировать изменения, внесенные вами в дочернюю ветку (в данном случае добавление нового файла), в основную ветку.

- В Child_Branch нажмите кнопку «Compare & pull request».



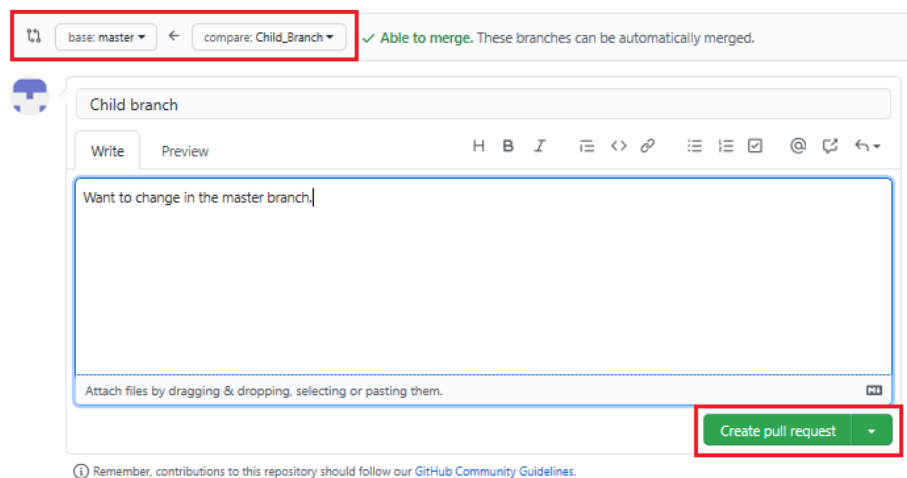
- Прокрутите страницу вниз и обратите внимание, что в списке указан *1 changed file*.



- Прокрутите страницу вверх и обратите внимание, что GitHub сравнивает ветки master и Child_Branch и между ними нет конфликтов. При желании вы можете добавить комментарий к запросу на включение. Нажмите *Create pull request*.

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).



4. Операция «Merge» на базе «pull request»

Чтобы объединить ветки по запросу pull request в проекте, выполните следующие действия:

- Откройте вкладку «*Pull requests*». Отображается список ожидающих запросов на включение.
- Щелкните pull request, который вы хотите объединить с основным проектом. Если вас устраивают изменения, нажмите «*Merge pull request*», чтобы принять запрос на включение и объединить обновления (при желании вы можете добавить комментарий).

Child branch #1

Open Malika-s wants to merge 3 commits into `master` from `Child_Branch`

Conversation 0 Commits 3 Checks 0 Files changed 1

Malika-s commented 2 minutes ago Owner

Want to change in the master branch.

Malika-s added 3 commits 25 minutes ago

- Create testchild Verified 9767921
- Delete testchild Verified 48b4479
- Create testchild.py Verified eb5ced2

Add more commits by pushing to the `Child_Branch` branch on Malika-s/testrepo.

Continuous integration has not been set up
GitHub Actions and several other apps can be used to automatically catch bugs and enforce style.

This branch has no conflicts with the base branch
Merging can be performed automatically.

Merge pull request You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

- При нажатии кнопки «Merge pull request» отображается кнопка «Confirm merge». Нажмите эту кнопку, чтобы завершить слияние.

Child branch #1

Open Malika-s wants to merge 3 commits into `master` from `Child_Branch`

Conversation 0 Commits 3 Checks 0 Files changed 1

Malika-s commented 4 minutes ago Owner

Want to change in the master branch.

Malika-s added 3 commits 27 minutes ago

- Create testchild Verified 9767921
- Delete testchild Verified 48b4479
- Create testchild.py Verified eb5ced2

Add more commits by pushing to the `Child_Branch` branch on Malika-s/testrepo.

Merge pull request #1 from Malika-s/Child_Branch

Child branch

Confirm merge Cancel

pull request теперь успешно merge. Обратите внимание: вы можете удалить дочернюю ветку, поскольку ваши изменения были включены в master ветку.

Branch: master ▾		Go to file	Add file ▾	Code ▾
Malika-s committed 7a7c576 3 minutes ago ... 8 commits 2 branches 0 tags				
README.md	Update README.md	3 hours ago		
Test.py	Add files via upload	1 hour ago		
firstpython.py	Create firstpython.py	3 hours ago		
testchild.py	Create testchild.py	21 minutes ago		

5. Работа с локальным репозиторием посредством командной строки.

Данный вариант доступен либо в MacOS / Linux, либо в Windows PowerShell через специальное приложение Git.

Создаем новую директорию для локального репозитория:

- Создайте каталог myrepo, скопировав и вставив в терминал приведенную ниже команду mkdir: «mkdir myrepo».
- Перейдите в каталог myrepo, скопировав и вставив команду cd: «cd myrepo».
- В этом каталоге myrepo можно создать новый локальный репозиторий git с помощью команды git init. Скопируйте и вставьте в терминал команду: «git init»
- Теперь создан новый локальный репозиторий, который вы можете проверить, выполнив список каталогов, вставив следующую команду в окно терминала: «ls -la .git»
- Вывод показывает содержимое подкаталога .git, в котором находится локальный репозиторий:

```

theia@theiadocker-rsahuja: /home/project/myrepo X
theia@theiadocker-rsahuja:/home/project$ mkdir myrepo
theia@theiadocker-rsahuja:/home/project$ cd myrepo
theia@theiadocker-rsahuja:/home/project/myrepo$ git init
Initialized empty Git repository in /home/project/myrepo/.git/
theia@theiadocker-rsahuja:/home/project/myrepo$ ls -la .git
total 40
drwxr-sr-x 7 theia users 4096 Jan 15 01:53 .
drwxr-sr-x 3 theia users 4096 Jan 15 01:53 ..
drwxr-sr-x 2 theia users 4096 Jan 15 01:53 branches
-rw-r--r-- 1 theia users  92 Jan 15 01:53 config
-rw-r--r-- 1 theia users  73 Jan 15 01:53 description
-rw-r--r-- 1 theia users  23 Jan 15 01:53 HEAD
drwxr-sr-x 2 theia users 4096 Jan 15 01:53 hooks
drwxr-sr-x 2 theia users 4096 Jan 15 01:53 info
drwxr-sr-x 4 theia users 4096 Jan 15 01:53 objects
drwxr-sr-x 4 theia users 4096 Jan 15 01:53 refs
theia@theiadocker-rsahuja:/home/project/myrepo$

```

- Теперь давайте создадим пустой файл, используя следующую команду: «touch newfile».
- Добавьте этот файл в репозиторий, используя следующую команду git add: «git add newfile»
- Прежде чем мы сможем зафиксировать наши изменения, нам нужно сообщить git, кто мы такие. Мы можем сделать это, используя следующие команды (вы можете скопировать эти команды как есть, без необходимости вводить фактическую информацию):

«git config --global user.email "you@example.com"»

«git config --global user.name "Your Name"»

- Как только в репозитории появится newfile, давайте зафиксируем наши изменения, используя команду git commit. Обратите внимание, что для фиксации требуется сообщение, которое мы включаем с помощью параметра -m: «git commit -m "added newfile"»

```
theia@theiadocker-rsahuja:/home/project/myrepo$ git commit -m "added newfile"
[master (root-commit) 161ac8d] added newfile
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 newfile
theia@theiadocker-rsahuja:/home/project/myrepo$
```

- Наш предыдущий коммит создал основную ветку по умолчанию под названием master. Чтобы внести последующие изменения в наш репозиторий, давайте создадим новую ветку в нашем локальном репозитории. Скопируйте и вставьте следующую команду git branch в терминал, чтобы создать ветку с именем my1stbranch: «git branch my1stbranch»

- Давайте проверим, какие ветки содержит наше репо, вставив в терминал следующую команду git branch: «git branch».

```
theia@theiadocker-rsahuja:/home/project/myrepo$ git branch my1stbranch
theia@theiadocker-rsahuja:/home/project/myrepo$ git branch
* master
  my1stbranch
theia@theiadocker-rsahuja:/home/project/myrepo$
```

Обратите внимание, что в выводе перечислены две ветки – основная ветка по умолчанию со звездочкой * рядом с ней, указывающая, что это активная в данный момент ветка, и вновь созданная ветка my1stbranch.

- Поскольку теперь мы хотим работать в новой ветке, введите команду git checkout, чтобы сделать ее активной веткой для внесения изменений: «git checkout my1stbranch».

- Давайте проверим, что новая ветка теперь является активной, выполнив следующую команду git branch: «git branch»

```
theia@theiadocker-rsahuja:/home/project/myrepo$ git branch my1stbranch
theia@theiadocker-rsahuja:/home/project/myrepo$ git branch
* master
  my1stbranch
theia@theiadocker-rsahuja:/home/project/myrepo$ git checkout my1stbranch
Switched to branch 'my1stbranch'
theia@theiadocker-rsahuja:/home/project/myrepo$ git branch
  master
* my1stbranch
theia@theiadocker-rsahuja:/home/project/myrepo$
```

Обратите внимание, что звездочка * теперь находится рядом с my1stbranch, указывая, что она теперь активна.

ПРИМЕЧАНИЕ. В качестве команды для создания новой ветки и перехода к ней с помощью git, вы можете также использовать следующую команду с опцией -b: «git checkout -b my1stbranch»

- Давайте внесем некоторые изменения в вашу новую ветку под названием my1stbranch. Начните с добавления текста в newfile, вставив в терминал следующую команду, которая добавит строку «Вот текст в моем новом файле» в файл: «echo 'Here is some text in my newfile.' >> newfile»

- Убедитесь, что текст добавлен, вставив следующую команду cat: «cat newfile»


```
theia@theiadocker-rsahuja:/home/project/myrepo$ echo 'Here is some text in my newfile.' >> newfile
theia@theiadocker-rsahuja:/home/project/myrepo$ cat newfile
Here is some text in my newfile.
```

- Теперь давайте создадим еще один файл с именем `readme.md`, используя следующую команду: «`touch readme.md`»

- Добавьте его в репозиторий с помощью следующей команды `git add`: «`git add readme.md`»

На данный момент в нашей новой ветке мы отредактировали новый файл и добавили файл с именем `readme.md`. Мы можем легко проверить изменения в нашей текущей ветке с помощью команды `git status`. Вывод команды `git status` показывает, что файлы `readme.md` были добавлены в ветку и готовы к фиксации, поскольку мы добавили их в ветку с помощью `git add`. Однако, несмотря на то, что мы изменили файл с именем `newfile`, мы не добавили его явно с помощью `git add`, и, следовательно, он не готов к фиксации:

```
theia@theiadocker-rsahuja:/home/project/myrepo$ touch readme.md
theia@theiadocker-rsahuja:/home/project/myrepo$ git add readme.md
theia@theiadocker-rsahuja:/home/project/myrepo$ git status
On branch my1stbranch
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   readme.md

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:   newfile
```

Команда для добавления всех изменений и дополнений – `git add` со звездочкой `*`; данный шаг также добавит измененный файл `newfile` в ветку и подготовит его к фиксации: «`git add *`»

- Давайте еще раз проверим статус: «`git status`». Вывод теперь показывает, что оба файла теперь могут быть зафиксированы:

```
theia@theiadocker-rsahuja:/home/project/myrepo$ git add *
theia@theiadocker-rsahuja:/home/project/myrepo$ git status
On branch my1stbranch
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   newfile
        new file:   readme.md
```

- Теперь, когда наши изменения готовы, мы можем сохранить их в ветку, используя следующую команду фиксации с сообщением, указывающим изменения: «`git commit -m "added readme.md modified newfile"`»

- Мы можем выполнить следующую команду `git log`, чтобы получить историю последних коммитов. В журнале показаны два недавних коммита — последний коммит в `my1stbranch`, а также предыдущий коммит в `master`.


```

theia@theiadocker-rsahuja:/home/project/myrepo$ git log
commit 9eb37c754d77231a2013781aa5215f71040975ed (HEAD -> my1stbranch)
Author: Your Name <you@example.com>
Date: Sat Jan 15 04:00:50 2022 +0000

    added readme.md modified newfile

commit 161ac8d957bd0d904c85ef8883b36c5824f10d85 (master)
Author: Your Name <you@example.com>
Date: Sat Jan 15 02:07:41 2022 +0000

    added newfile

```

* Чтобы выйти из журнала git, введите q.

- Иногда вы не можете полностью протестировать свои изменения перед их фиксацией, и это может иметь нежелательные последствия. Вы можете отменить свои изменения, используя команду `git revert`, как показано ниже. Вы можете либо указать идентификатор вашего коммита, который вы можете увидеть в предыдущем выводе журнала, либо использовать ярлык HEAD для отката последнего коммита:

«`git revert HEAD --no-edit`»

Если вы не укажете флаг `--no-edit`, вам может быть представлен экран редактора, показывающий сообщение с изменениями, которые необходимо отменить. В этом случае нажмите клавишу Control (или Ctrl) одновременно с X. Вывод показывает, что самый последний коммит с указанным идентификатором был отменен:

```

theia@theiadocker-rsahuja:/home/project/myrepo$ git revert HEAD --no-edit
[my1stbranch f4f5600] Revert "modified newfile added readme.md"
Date: Sat Jan 15 04:39:38 2022 +0000
2 files changed, 1 deletion(-)
delete mode 100644 readme.md
theia@theiadocker-rsahuja:/home/project/myrepo$

```

- Давайте внесем еще одно изменение в ваш текущий активный my1stbranch, используя следующие команды:

«`touch goodfile`»

«`git add goodfile`»

«`git commit -m "added goodfile"`»

«`git log`»

- Вывод журнала показывает, что недавно добавленный goodfile был зафиксирован в my1stbranch:

```

theia@theiadocker-rsahuja:/home/project/myrepo$ git status
On branch my1stbranch
nothing to commit, working tree clean
theia@theiadocker-rsahuja:/home/project/myrepo$ touch goodfile
theia@theiadocker-rsahuja:/home/project/myrepo$ git add goodfile
theia@theiadocker-rsahuja:/home/project/myrepo$ git commit -m "added goodfile"
[my1stbranch d8680b4] added goodfile
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 goodfile
theia@theiadocker-rsahuja:/home/project/myrepo$ git log
commit d8680b4cf63ff3e97b2970704806c14dc6134dd1 (HEAD -> my1stbranch)
Author: Your Name <you@example.com>
Date: Sat Jan 15 04:53:27 2022 +0000

    added goodfile

```

- Теперь давайте объединим содержимое my1stbranch с основной веткой. Сначала нам нужно сделать активную ветку master с помощью следующей команды git checkout: «git checkout master»

- Теперь давайте объединим изменения из my1stbranch в master.

«git merge my1stbranch»

«git log»

Вывод и журнал показывают успешное слияние ветки:

```
theia@theiadocker-rsahuja:/home/project/myrepo$ git checkout master
Switched to branch 'master'
theia@theiadocker-rsahuja:/home/project/myrepo$ git merge my1stbranch
Updating 8954f54..d8680b4
Fast-forward
 goodfile | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 goodfile
theia@theiadocker-rsahuja:/home/project/myrepo$ git log
commit d8680b4cf63ff3e97b2970704806c14dc6134dd1 (HEAD -> master, my1stbranch)
Author: Your Name <you@example.com>
Date: Sat Jan 15 04:53:27 2022 +0000

    added goodfile
```

- Теперь, когда изменения были объединены в главную ветку, my1stbranch можно удалить с помощью следующей команды git branch с опцией -d: «git branch -d my1stbranch»

```
theia@theiadocker-rsahuja:/home/project/myrepo$ git branch -d my1stbranch
Deleted branch my1stbranch (was d8680b4).
theia@theiadocker-rsahuja:/home/project/myrepo$ █
```

Задание №1:

- Создайте новую ветку под названием newbranch.
- Сделать новую ветку активной.
- Создайте пустой файл с именем newbranchfile.
- Добавьте вновь созданный файл в свою ветку.
- Зафиксируйте изменения в новой ветке.
- Отмените последние зафиксированные изменения.
- Создайте новый файл с именем newgoodfile.
- Добавьте последний файл в новую ветку.
- Зафиксируйте изменения.
- Объедините изменения в новой ветке с основной.

Задание №2:

Разбившись на подгруппы подвое выполнить

- fork проекта второго студента;
- clone проекта в локальный репозиторий;
- добавление и фиксирование файла;
- синхронизацию с fork-репозиторием (“git push --set-upstream origin ” + имя новой ветки, если таковая создавалась -> merge этих двух веток; если изменения выполнялись сразу в master branch, то используется простая команда «git push origin master»); для git push понадобится имя пользователя и не простой пароль пользователя, а token (classic), полученный в соответствии с инструкцией <https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-CD0131EN-SkillsNetwork/labs/create-personal-token/instructions.md.html> (вводится вместо пароля в ходе выполнения git push)
- формирование pull request к origin проекту на прием данного изменения.