# 10 простых шагов с Git.

Следуя инструкциям в «шагах» вам необходимо будет в конечном итоге загрузить на репозиторий проект.

## Шаг 1: Установка git и создание аккаунта на GitHub

Чтобы работать с git его надо для начала установить. Если у вас свой ноутбук и это ещё не было произведено, то перейдите [по ссылке](https://git-scm.com/downloads) и установите версию. В аудиторных ноутбуках это уже было произведено, но дома возможно вам это понадобиться. После этого вам потребуется создать аккаунт на GitHub, перейдите [по ссылке](https://github.com/join) и зарегистрируйтесь.

Небольшое отступление: **git** и **GitHub** — это не одно и то же. **Git** — это инструмент контроля версий с открытым исходным кодом, созданный в 2005 году разработчиками, работающими над операционной системой **Linux**.

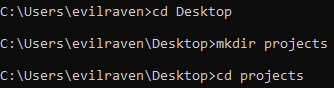
**GitHub** — компания, основанная в 2008 году, которая производит инструменты, интегрируемые с **git**. Вам не нужен **GitHub** для использования **git**, но вы не можете использовать **GitHub** без использования **git**.

Существует множество других альтернатив **GitHub**, таких как **GitLab**, **BitBucket**, а также решения «хост для себя», такие как **gogs** и **gittea**. Все это на языке **git** называется «удаленным доступом», и все они совершенно необязательны. Вам не нужно использовать удаленный доступ для использования **git**, но это облегчит обмен вашим кодом с другими.

## Шаг 2: Создание локального git репозитория

При создании нового проекта на локальном компьютере с помощью git вы сначала создаете [новый репозиторий](https://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository) (или, для краткости, часто «repo»).

Для работы с git, вы будем использовать терминал. Для начала откройте терминал и перейдите туда, где вы хотите разместить проект на локальном компьютере, с помощью команды cd (изменить каталог). Например, если у вас на рабочем столе есть папка «Проекты», вы должны сделать что-то вроде:



Чтобы инициализировать репозиторий git в корне папки, выполните команду **git init**:



## Шаг 3. Добавление новых файлов в репозиторий.

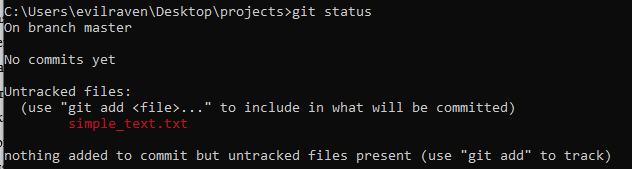
Добавьте в проект новый текстовый файл командой

echo текст\_для\_файла > имя\_файла.расширение\_файла



После того как вы добавили или изменили файлы в папке, содержащей репозиторий **git**, **git** заметит, что файл существует внутри репозитория. Но **git** не будет отслеживать файл, если вы явно не укажете это. **Git** сохраняет/управляет изменениями только в файлах, которые он отслеживает, поэтому нам нужно будет отправить команду, чтобы подтвердить, что да, мы хотим, чтобы **git** отслеживал наш новый файл.

После создания нового файла вы можете использовать команду **git status**, чтобы увидеть, какие файлы, по сведениям **git**, существуют.



По сути, это означает следующее: «Эй, мы заметили, что вы создали новый файл с именем simple\_text.txt, но если вы не введете команду «git add», с этим файлов не будет ничего происходить».

Небольшое отступление: промежуточная среда, коммит и вы

Одна из самых запутанных частей при первом изучении **git** — это концепция промежуточной среды и ее отношение к коммиту. **Коммит** — это запись о том, какие изменения вы внесли с момента последнего коммита. По сути, вы вносите изменения в свое repo (например, добавление файла или его изменение), а затем сообщили в **gitе** поместить эти изменения в коммит. Коммиты составляют суть вашего проекта и позволяют вам перейти к состоянию проекта при любом другом коммите.

И так, как же указать **git**, какие файлы следует поместить в коммит? Здесь на помощь приходит [промежуточная среда или индекс](https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository). Как видно из шага 2, когда вы вносите изменения в свой репозиторий, **git** замечает, что файл изменился, но ничего с ним не делает (например, добавляет его в коммит).

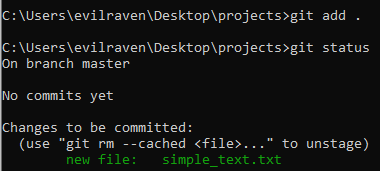
Чтобы добавить файл в коммит, сначала необходимо добавить его в промежуточную среду. Для этого вы можете использовать команду **git add «filename»**.

После того, как вы использовали команду **git add** для добавления всех нужных файлов в промежуточную среду, вы можете указать **git** упаковать их в коммит с помощью команды **git commit**.

## Шаг 4: Добавление файла в промежуточную среду

Добавьте файл в промежуточную среду с помощью команды **git add «filename»** или можно добавить все файлы сразу же командой **git add «знак точки»**.

Если вы повторно запустите команду **git status**, вы увидите, что **git** добавил файл в промежуточную среду (обратите внимание на строку «Changes to be committed»).



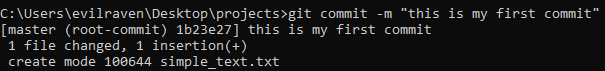
Еще раз повторяю: файл еще не добавлен в коммит, но скоро будет добавлен.

## Шаг 5: Создание коммита

Пришло время создать свой первый коммит!

Запустите команду **git commit -m "Ваше сообщение о коммите"** (-m это сокращение от message)

(Возможно при первом коммите, если не разу не был отправлен коммит, то надо бы там оставить свои первоначальные данные: **git config --global user.name "nameuser"**; **git config --global user.email "useremail"**)



Сообщение в конце коммита должно быть связано с тем, что содержит коммит — может быть, это новая функция, может быть, это исправление ошибки, а может быть, это просто исправление опечатки. **Не размещайте сообщения типа «asdfadsf» или «prikol». Это вводит в диссонанс других людей, которые видят ваши коммиты в процессе разработки. Никаких «лол, кек, чебурек».**

Коммиты живут в репозитории вечно (технически вы можете удалить их, если вам это действительно нужно, но это беспорядочно), поэтому, если вы оставите четкое объяснение своих изменений, это может быть чрезвычайно полезно для будущих программистов (возможно, будущих вас!), которые пытаясь выяснить, почему годы спустя были внесены некоторые изменения.

## Шаг 6: Создание новой ветки

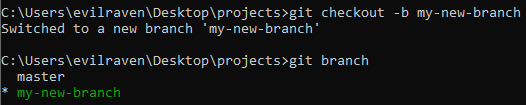
Теперь, когда вы сделали новый коммит, давайте попробуем что-нибудь более продвинутое.

Предположим, вы хотите создать новую функцию, но беспокоитесь о внесении изменений в основной проект во время разработки этой функции. Здесь на помощь приходят [ветки git](https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell).

Ветки позволяют вам перемещаться вперед и назад между «состояниями» проекта. Официальная документация **git** описывает ветки следующим образом: «Ветка в **Git** — это просто легкий подвижный указатель на один из этих коммитов». Например, если вы хотите добавить новую страницу на свой веб-сайт, вы можете создать новую ветку только для этой страницы, не затрагивая основную часть проекта. Закончив работу со страницей, вы можете [объединить изменения](https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging) из своей ветки с основной веткой. Когда вы создаете новую ветку, **Git** отслеживает, от какого коммита «разветвилась» ваша ветка, поэтому он знает историю всех файлов.

Допустим, вы находитесь в основной ветке и хотите создать новую ветку для разработки своей веб-страницы. Вот что вам нужно сделать: введите команду [**git checkout**](https://git-scm.com/docs/git-checkout) **-b «имя моей ветки»**. Эта команда автоматически создаст новую ветку, а затем перемещает вас в новую созданную ветку.

После запуска приведенной выше команды вы можете использовать команду [**git branch**](https://git-scm.com/docs/git-branch), чтобы просмотреть ветки и убедиться, что ваша ветка была создана:



Название ветки со звездочкой рядом с ней указывает, в какой ветке вы находитесь в данный момент.

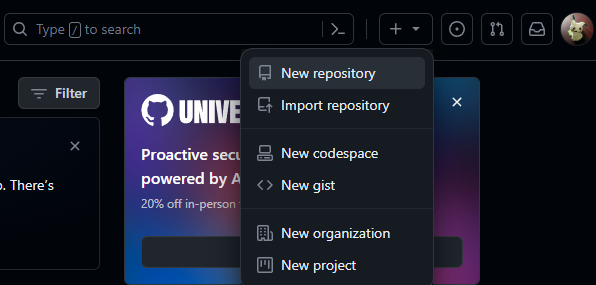
Небольшое отступление: примечание к названию основной ветвки

По умолчанию первая ветка каждого репозитория **git называется «master»** (и обычно используется в качестве основной ветки в проекте). В рамках общей работы в сфере высоких технологий некоторые группы начали использовать альтернативные названия ветки по умолчанию. В другой документации вы можете увидеть «main» или другие термины, используемые для обозначения основной ветки. Независимо от названия, просто имейте в виду, что почти у каждого репозитория есть основная ветка, которую можно рассматривать как официальную версию репозитория. Если это веб-сайт, то основной веткой является версия, которую видят пользователи. Если это приложение, тогда основной веткой будет версия, которую загружают пользователи.

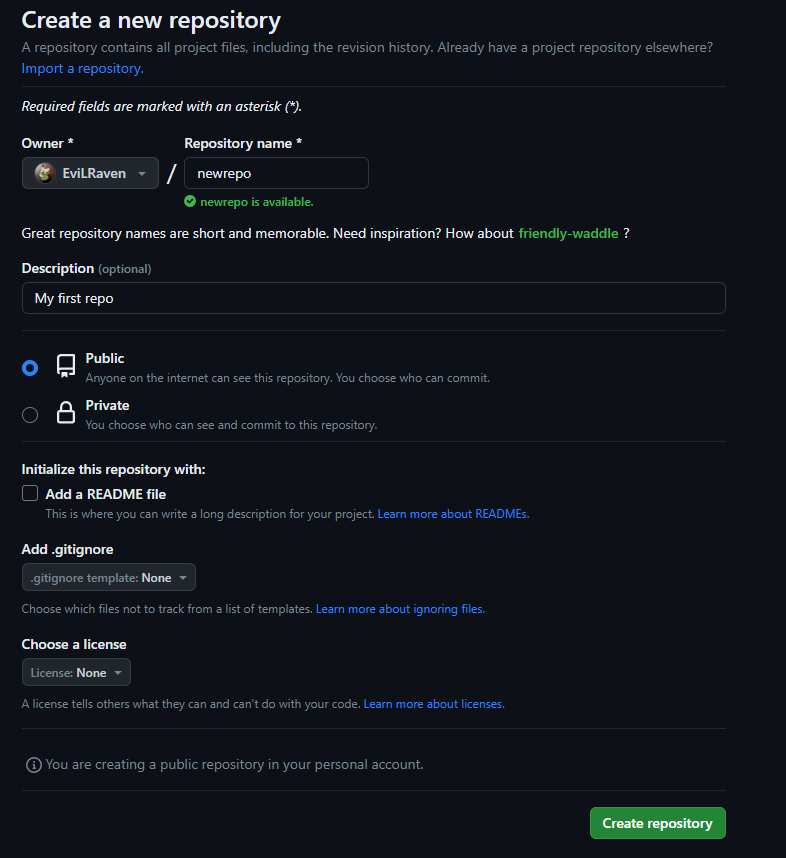
## Шаг 7: Создание репозитория на GitHub

Если вы хотите отслеживать свой код только локально, вам не нужно использовать GitHub. Но если вы хотите работать в команде, вы можете использовать GitHub для совместного изменения кода проекта.

Чтобы создать новый репозиторий на GitHub, войдите в систему и перейдите на домашнюю страницу GitHub. Вы можете найти опцию «Новый репозиторий» под знаком «+» рядом с изображением вашего профиля в правом верхнем углу навигационной панели:

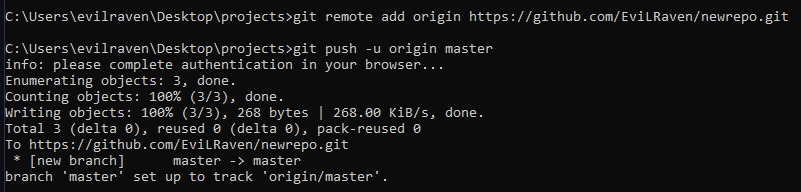


После нажатия кнопки GitHub попросит вас назвать репозиторий и предоставить краткое описание



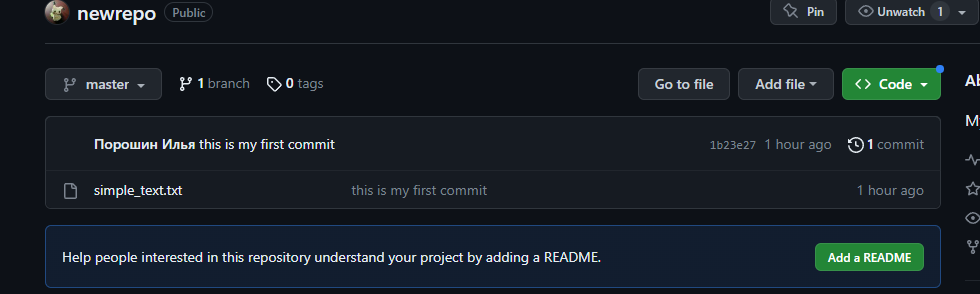
Когда вы закончите заполнять информацию, нажмите кнопку «Создать репозиторий», чтобы создать новый репозиторий.

После нажатия кнопки, необходимо будет выгрузить локальный созданный проект на GitHub репозиторий. Для этого скопируйте адрес репозитория и введите команды в консоли.



После введения команды git push -u origin master вас попросят авторизоваться в экосистеме GitHub. Вы можете выбрать «войти через браузер» или же, войти по токену, созданному в профиле GitHub.

Обновите страницу на репозитории в GitHub и вы увидите загруженный репозиторий, поздравляю!

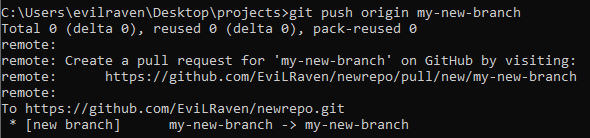


(Заметьте, что выгружена лишь одна ветка master)

## Шаг 8: Выгрузка веток в GitHub

Теперь мы отправим коммит из вашей ветки в ваш новый репозиторий **GitHub**. Это позволит другим людям видеть внесенные вами изменения. Если они одобрены владельцем репозитория, изменения можно затем объединить с основной веткой.

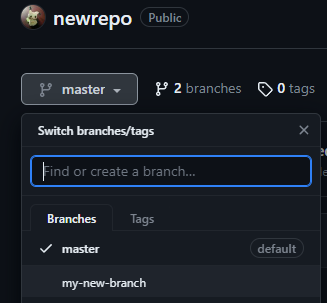
Чтобы отправить изменения в новую ветку на **GitHub**, вам нужно ввести команду [**git push**](https://git-scm.com/docs/git-push) **origin yourbranchname**. **GitHub** автоматически создаст для вас ветку в удаленном репозитории



Возможно, вам интересно, что означает слово «origin» в приведенной выше команде. Происходит следующее: когда вы клонируете удаленный репозиторий на свой локальный компьютер, git создает для вас псевдоним. Почти во всех случаях этот псевдоним называется «origin». По сути, это сокращение URL-адреса удаленного репозитория. И так, чтобы отправить изменения в удаленный репозиторий, вы могли бы использовать либо команду: **git push git@github.com:git/git.git yourbranchname** или **git push origin yourbranchname**

(Если вы впервые используете GitHub локально, вам может быть предложено войти в систему, используя имя пользователя и пароль GitHub.)

Если вы обновите страницу GitHub, вы увидите сообщение о том, что ветка с вашим именем только что была добавлена в репозиторий. Вы также можете нажать ссылку «branches», чтобы увидеть там свою ветку.



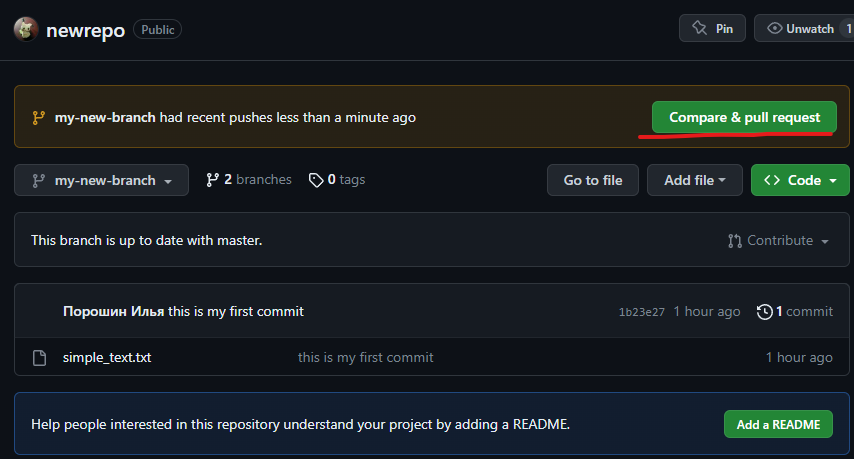
Следующим шагом мы должны будем уведомить (pull request) хозяина репозитория об объединении веток.

## Шаг 9: Создание pull request (PR) и объединение веток

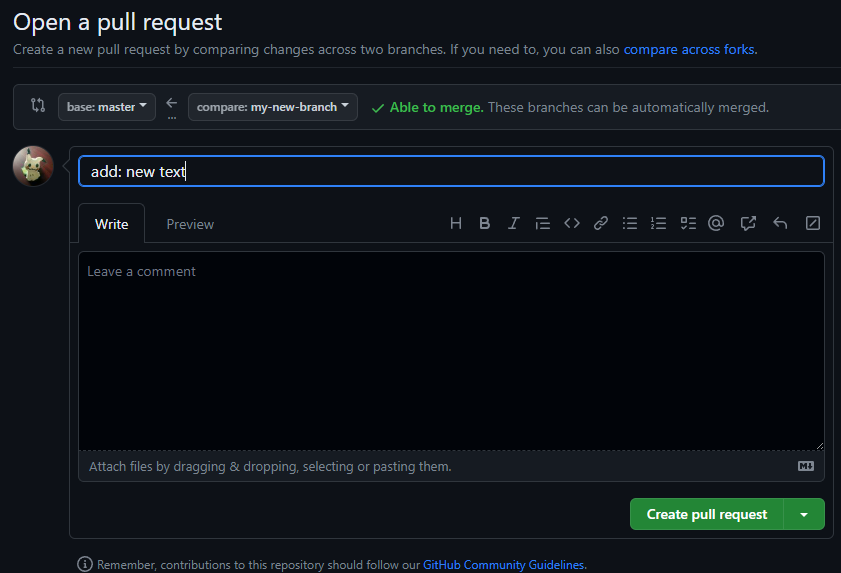
Pull request (или PR) — это способ предупредить владельцев репозитория о том, что вы хотите внести некоторые изменения в код. Это позволяет просмотреть код и убедиться, что он выглядит хорошо, прежде чем вносить изменения в основную ветку.

Добавьте новый файл в новую ветку и выгрузите в репозиторий.

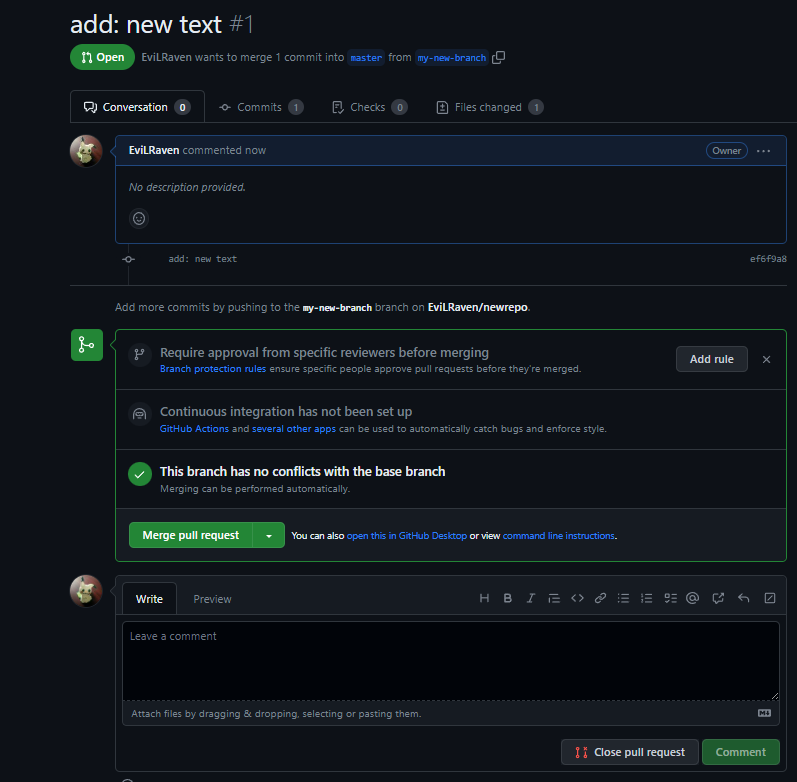
Откройте страницу репозитория вы увидите сообщение о PR.



Вот как выглядит PR-страница до ее отправки:

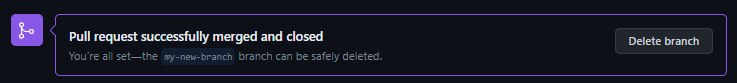


Вот как это выглядит после отправки запроса на PR:

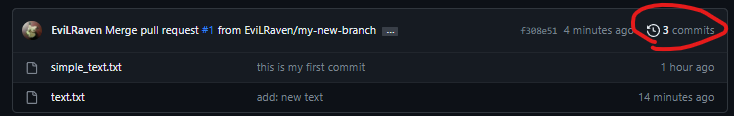


Внизу вы можете увидеть большую зеленую кнопку с надписью «Объединить запрос». Нажатие на эту кнопку означает, что вы объедините свои изменения с основной веткой. Нажмите на кнопку зеленую «Merge pull request» для объединения веток.

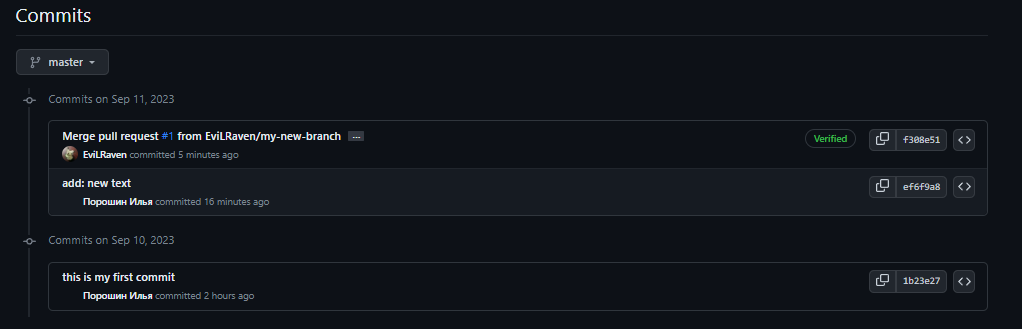
После объединения веток, вы можете удалить с удаленного репозитория не используемую ветку.



Вы можете дважды проверить, были ли ваши коммиты объединены, нажав ссылку «Коммиты» на первой странице вашего нового репозитория.



Это покажет вам список всех коммитов в этой ветке. Вы можете увидеть тот, который я только что объединил, вверху (merge pull request № 1).

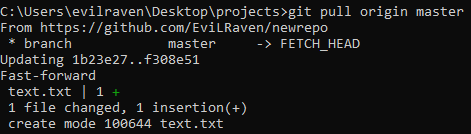


Вы также можете увидеть [хэш-код](https://git-scm.com/docs/git-hash-object) коммита справа. Хэш-код — это уникальный идентификатор конкретного коммита. Это полезно для ссылки на определенные коммиты и при отмене изменений (используйте команду [git revert](https://git-scm.com/docs/git-revert) «номер хэш-кода» для возврата).

## Шаг 10: Получение изменений с GitHub обратно на свой компьютер.

Прямо сейчас репозиторий на GitHub выглядит немного иначе, чем на вашем локальном компьютере. Например, коммит, который вы сделали в своей ветке и объединили с основной веткой, не существует в основной ветке на вашем локальном компьютере.

Чтобы получить самые последние изменения, которые вы или другие пользователи объединили на GitHub, используйте команду **git pull origin namebranch** (вам необходимо будет обновить main ветку). В большинстве случаев это можно сократить до «git pull». Перейдите в ветку мастер с помощью команды git checkout master и обновите её.



Здесь показаны все файлы, которые были изменены, и то, как они изменились.

Теперь мы можем снова использовать команду [**git log**](https://git-scm.com/docs/git-log), чтобы увидеть все новые коммиты.

