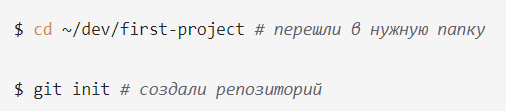
# **Инициализируем репозиторий**

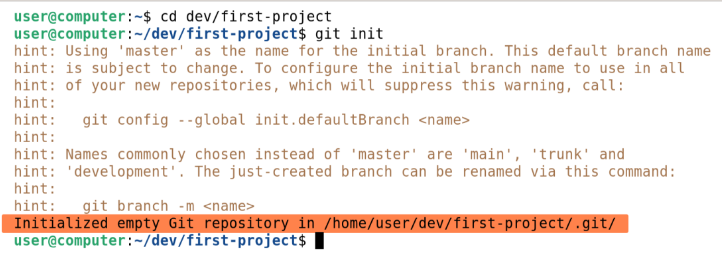
# **Сделать папку репозиторием — git init**

Чтобы Git начал отслеживать изменения в проекте, папку с файлами этого проекта нужно сделать **Git-репозиторием** (от англ. *repository* — «хранилище»). Для этого следует переместиться в неё и ввести команду git init (от англ. ***init****ialize* — «инициализировать»).



Вы можете создать папку в любом месте на компьютере. Но в этом случае не забывайте менять в наших примерах путь на тот, который ведёт к вашей папке. Помните, что не рекомендуется создавать репозиторий Git внутри другого Git-репозитория. Это может вызывать проблемы с отслеживанием изменений. В некоторых случаях при инициализации репозитория Git может показать объёмное сообщение, которое начинается со слов Using 'master' as the name…. Не пугайтесь: это не ошибка. Пока это сообщение не имеет большого значения.

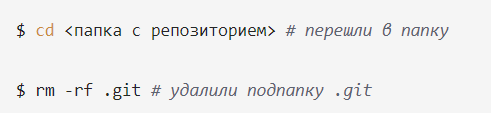
Также git init выведет сообщение вида Initialized empty Git repository in <\*ваша папка с проектом\*>/.git/ (англ. «инициализирован пустой Git-репозиторий в <\*ваша папка\*>/.git/»). В подпапке .git Git будет хранить всю служебную информацию.



Команда git init — одна из редко применяемых, ведь репозиторий создаётся один раз, а пользоваться им можно сколько угодно долго.

### **«Разгитить» папку, если что-то пошло не так, — rm -rf .git**

Если вы случайно сделали Git-репозиторием не ту папку, её можно «разгитить». Для этого нужно удалить скрытую подпапку .git.

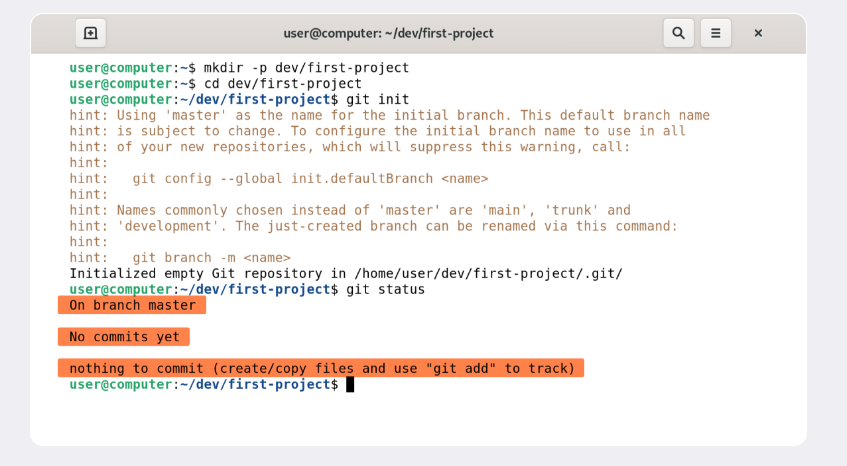


* ключ -r (от англ. ***r****ecursive* — «рекурсивно») позволяет удалять папки вместе с их содержимым;
* ключ -f (от англ. ***f****orce* — «заставить») избавит вас от вопросов вроде «Вы точно хотите удалить этот файл? А этот? И этот тоже?».

В подпапке .git хранится история изменений. Если удалить .git, то вся история проекта будет стёрта без возможности восстановления — останется только последняя версия файлов.

### **Проверить состояние репозитория — git status**

После инициализации репозитория first-project запустите команду git status (от англ. *status* — «статус», «состояние») — она показывает текущее состояние репозитория.



Команда git status выведет:

* название текущей ветки: On branch master или On branch main;
* сообщение о том, что в репозитории ещё нет коммитов: No commits yet;
* сообщение, которое говорит: «чтобы что-нибудь закоммитить (то есть зафиксировать), нужно сначала это создать» — nothing to commit (create/copy files and use "git add" to track).

# **Добавляем файлы в репозиторий**

Вы инициализировали Git-репозиторий, но в нём пока ничего нет. В этом уроке разберём, как добавить туда файлы.

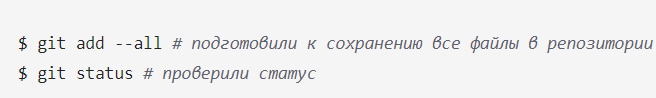
### **Подготовить файлы к сохранению — git add**

Добавим в репозиторий два файла. Например, файл todo.txt, в котором будет список дел, и readme.txt для информации о проекте. git status сообщит, что в папке first-project есть untracked files (от англ. *track* — «следить», *untracked* — «неотслеженный», «неотслеживаемый») — ещё не отслеживаемые файлы readme.txt и todo.txt.

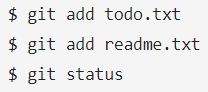


Состояние untracked значит, что Git ещё не хранит информацию о версиях файла и не может отследить, как он изменялся.

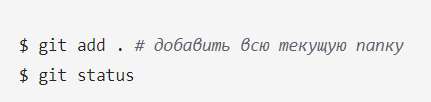
Сейчас в first-project два файла. Мы хотим отслеживать состояние обоих, поэтому можем использовать команду git add --all (от англ. *add* — «добавить» + от англ. *all* — «всё»). Ключ, или флаг, --all позволяет подготовить к сохранению все файлы в репозитории.



Добавлять файлы можно и по одному, без ключа --all.



Также можно добавить текущую папку целиком — в этом случае все файлы в ней тоже будут добавлены. Обратиться к текущей папке в Bash позволяет точка (.).



Вы можете использовать любой из этих вариантов — результат будет одинаковый.



# **Делаем первый коммит**

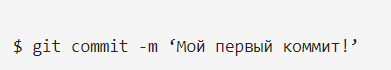
Коммит — это одна из основных сущностей в Git (и в других системах контроля версий). Коммит гарантирует, что изменения будут сохранены в истории и при необходимости к ним можно будет «откатиться». Это как если бы вы могли выполнить операцию Ctrl+Z для целой папки (репозитория).

**Выполнить коммит — git commit**

Сделать коммит можно командой git commit c ключом -m (от англ. ***m****essage* — «сообщение»), который присваивает коммиту сообщение.

Обычно в таком сообщении поясняется, в чём именно состояли изменения. Это как заметки на полях: благодаря им проще читать и понимать текст. Сообщение коммита выполняет те же функции — улучшает понимание и упрощает навигацию. Оно пишется после ключа -m в кавычках.

Например, перейдите в папку first-project и выполните коммит со следующим комментарием.



После нажатия Enter текущая версия файлов будет сохранена в репозитории с сообщением Мой первый коммит!. **Коммит** (по названию команды git commit) — это по сути список файлов с их контентом.



Команда git commit выведет информацию о коммите.

* [master (root-commit) baa3b6e] значит:
* коммит был в ветке master;
* root-commit — это самый первый, или «корневой» (англ. *root*), коммит в ветке, у следующих коммитов такой надписи не будет;
* baa3b6e — сокращённый идентификатор коммита (подробнее об этом мы ещё расскажем).
* 2 files changed, 1 insertion(+) значит:
* изменились два файла (readme.txt и todo.txt);
* одна строка была добавлена (1. Пройти пару уроков по Git.).
* Строки вида create mode 100644 readme.txt — это более подробная информация о новых (добавленных в Git) файлах.
* create (англ. «создать») говорит, что файл был создан. Если бы файл был удалён, на этом месте было бы слово delete (англ. «удалить»).
* mode 100644 сообщает, что это обычный файл. Также возможны варианты 100755 для исполняемых файлов (например, что-нибудь.exe) и 120000 для файлов-ссылок в Linux. Файлы-ссылки не содержат данных сами по себе, а только ссылаются на другие файлы — как «ярлыки» в Windows.

# **Просматриваем историю коммитов**

# **Просмотреть историю коммитов — git log**

В самостоятельном задании прошлого урока вы сделали три коммита в ваш репозиторий. Чтобы увидеть их все, введите команду git log (от англ. *log* — «журнал [записей]»).



Обратите внимание, что по умолчанию git log выводит коммиты в обратном хронологическом порядке — последние коммиты оказываются первыми сверху. В этом можно убедиться, если посмотреть на дату и время их создания.

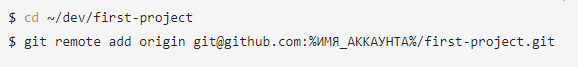
Если после выполнения команды вы видите, что в репозитории есть только один коммит или их нет вообще, вернитесь к прошлому уроку и убедитесь, что git add и git commit были вызваны в нужном порядке.

### **Привязать удалённый репозиторий к локальному — git remote add**

Перейдите на страницу удалённого репозитория, выберите тип SSH и скопируйте URL. Кнопка справа позволит сделать это мгновенно.



Откройте консоль, перейдите в каталог локального репозитория и введите команду git remote add (от англ. *remote* — «удалённый» и *add* — «добавить»).

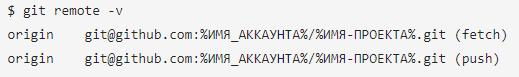


Команде необходимо передать два параметра: имя удалённого репозитория и его URL. В качестве имени используйте слово origin. А URL вы скопировали со страницы удалённого репозитория.

origin (англ. «источник») — стандартный псевдоним, с помощью которого можно обращаться к главному удалённому репозиторию (обычно такой репозиторий один). Это значительно упрощает работу.

### **Убедиться, что репозитории связаны, — git remote -v**

Отлично: вы связали локальный репозиторий с удалённым. Осталось убедиться, что всё работает, с помощью следующей команды.



В выводе вы должны увидеть две строчки, аналогичные тем, что показаны выше.

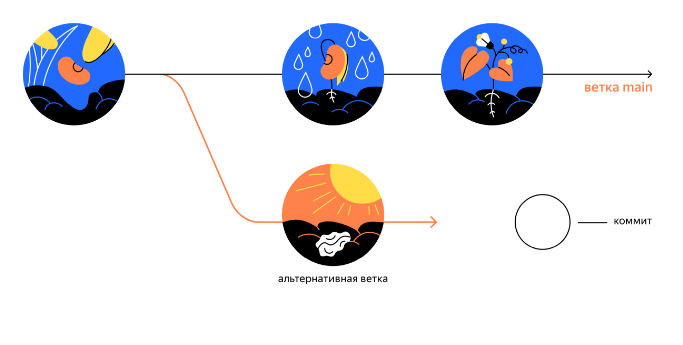
Флаг -v — короткая форма флага --verbose (англ. «подробный»). Он позволяет показать больше информации в выводе.

### **Основная ветка**

Мы упоминали, что каждый коммит сохраняет актуальное состояние файлов. Сами же коммиты хранятся в **ветках** (англ. *branch*).

Если коммит — это снимок состояния файлов, то ветка — временна́я шкала, на которой расположены эти снимки. Ветка всегда начинается от одного из коммитов.

В репозитории может существовать сразу несколько веток — параллельных историй изменений. Также они могут соединяться друг с другом.



Самая первая ветка в репозитории появляется автоматически и называется main (англ. «основная») или master. Её имя нужно указывать при отправке коммитов на удалённый репозиторий или при получении их из него.

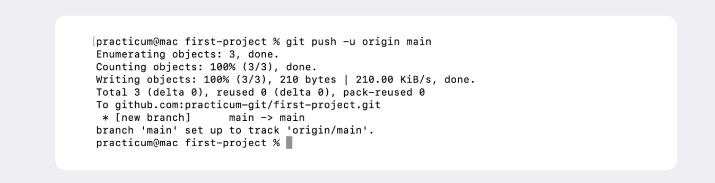
### **Отправить изменения на удалённый репозиторий — git push**

Вы уже прошли весь «цикл коммита»: подготовили файлы с помощью git add, закоммитили их с комментарием командой git commit -m. Осталось загрузить содержимое локального репозитория на GitHub. За это отвечает команда git push (от англ. *push* — «толкать»).

В первый раз эту команду нужно вызвать с флагом -u и параметрами origin (имя удалённого репозитория) и main или master (название текущей ветки). Флаг -u свяжет локальную ветку с одноимённой удалённой.



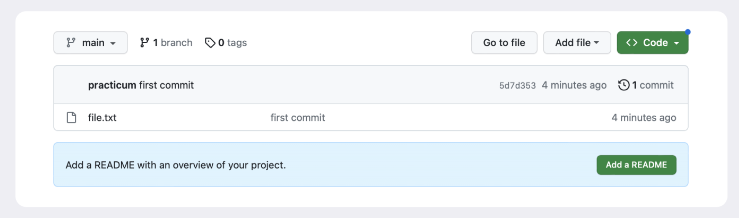
Появится такой экран.



отладочную информацию: количество объектов (файлов), которые отправляются на сервер, информацию о прогрессе сжатия и записи и так далее.

Если вы указывали кодовую фразу при настройке SSH-ключей, её нужно будет ввести.

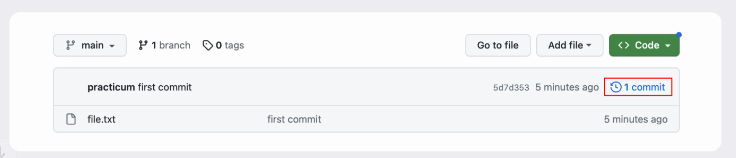
Зайдите в репозиторий first-project на GitHub. Вы увидите, что в репозитории появились файлы с изменениями.



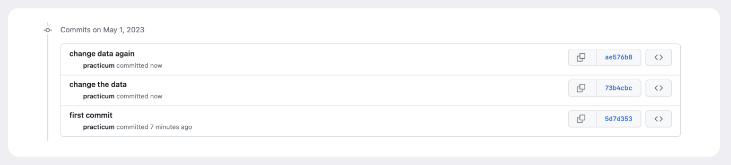
В дальнейшем при работе с удалённым репозиторием флаг -u можно опустить и писать просто git push.

### **Работа с графическим интерфейсом GitHub**

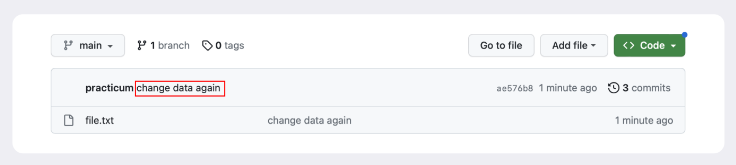
GitHub предоставляет удобный интерфейс для работы с репозиторием. Например, нажмите на кнопку **commit** в правой части страницы, чтобы просмотреть все коммиты в репозитории.



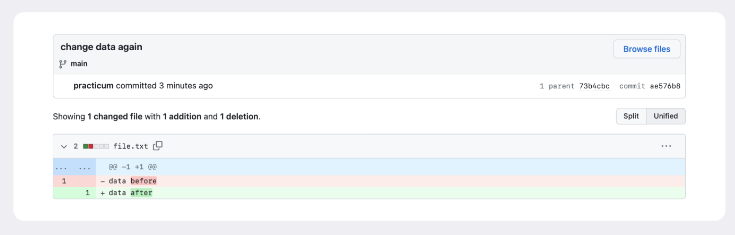
Откроется окно с коммитами и их авторами.



Сообщение коммита в репозитории тоже является ссылкой.



Перейдите по ссылке, кликните на текст последнего коммита над репозиторием — так вы сможете увидеть все изменения, которые были внесены в репозиторий в этом коммите.



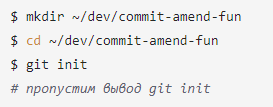
# **Как исправить коммит**

Иногда в только что выполненном коммите нужно что-то поменять: например, добавить ещё пару файлов или заменить сообщение на более информативное.

В таком случае можно внести правки в уже сделанный коммит с помощью опции --amend (от англ. *amend —* «исправить», «дополнить») у команды commit: git commit --amend. Разберём, как она работает.

### **Подготавливаем репозиторий**

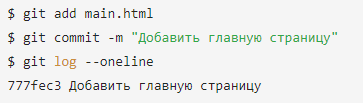
Создайте тренировочный репозиторий для отработки команды.



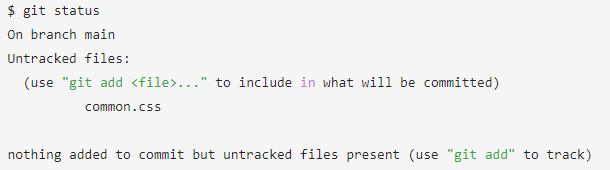
### **Дополнить коммит новыми файлами — git commit --amend --no-edit**

Представьте, что делаете небольшой сайт и для этого создали файл-страницу main.html, а также файл со стилями common.css.

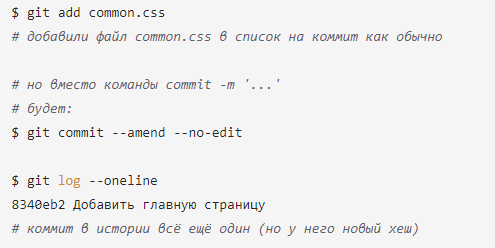
В какой-то момент вы забыли о файле common.css и добавили в коммит только main.html.



Файл common.css так и остался «висеть» в untracked. В этом легко убедиться, если вызвать git status.



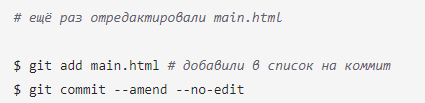
Дополните последний коммит забытым файлом common.css с помощью опции --amend.



С опцией --amend команда commit не создаст новый коммит, а дополнит последний, просто добавив в него файл common.css. При этом хеш последнего коммита изменится, потому что изменился список файлов в коммите.

Обратите внимание на опцию --no-edit. Она сообщает команде commit, что сообщение коммита нужно оставить как было.

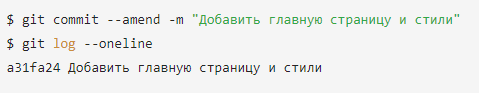
Точно так же можно добавить не новый файл, а дополнительные изменения в уже добавленном в коммит файле.



### **Изменить сообщение коммита — git commit --amend -m "Новое сообщение"**

Может быть и так, что добавлять новые файлы в коммит не нужно, зато понадобилось изменить сообщение.

Допустим, хочется заменить сообщение Добавить главную страницу на Добавить главную страницу и стили. Сделать это можно через commit --amend с флагом -m.



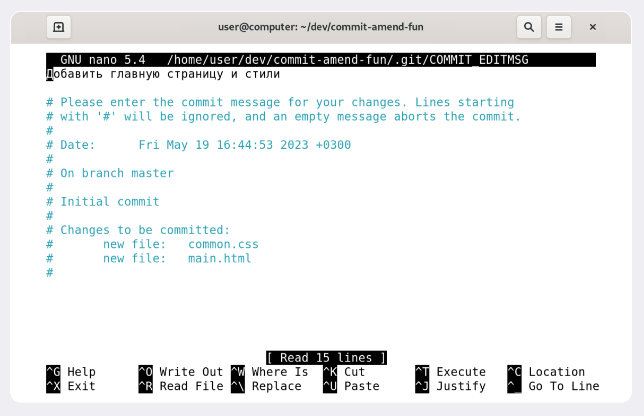
Хеш коммита снова поменялся, потому что изменились сообщение и время коммита. При этом файлы в коммите остались те же: main.html и common.css.

## **Случилось страшное: открылся редактор**

Если забыть указать у команды git commit --amend один из флагов (--no-edit или -m), Git предложит отредактировать сообщение коммита вручную. Для этого он откроет текстовый редактор, который установлен в системе по умолчанию. Чаще всего это либо **GNU nano**, либо **Vim**.

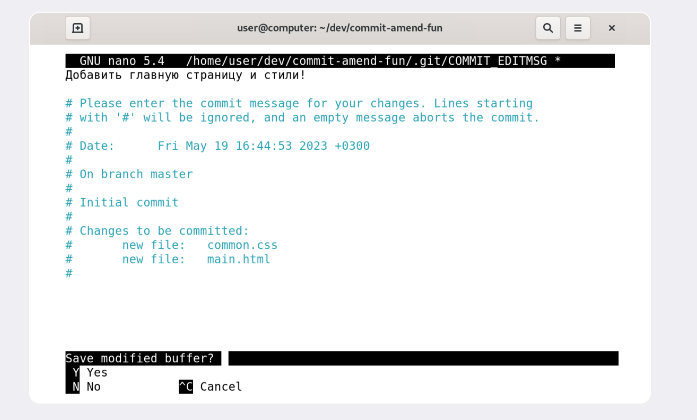
**nano** — простой и свободный

При редактировании сообщения коммита редактор GNU nano выглядит примерно так.



Первая строка Добавить главную страницу и стили — это текущее сообщение коммита. Если вы хотите изменить сообщение, нужно отредактировать эту строку.

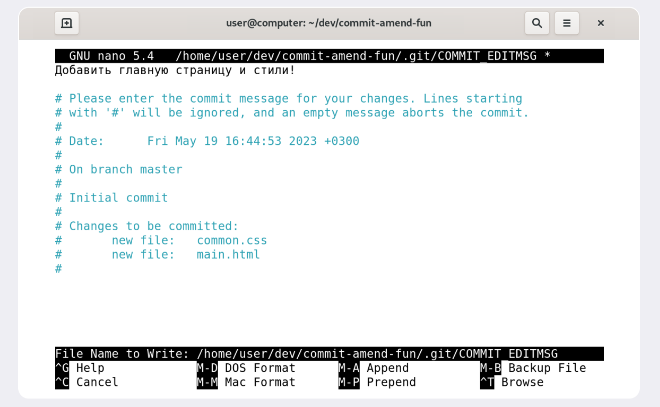
Допустим, решили добавить в конце сообщения восклицательный знак. Чтобы сохранить новое сообщение, нужно нажать Ctrl+X, где X значит *e****x****it* (англ. «выход»).



После нажатия Ctrl+X nano предложит сохранить файл, для этого нужно нажать Y (от англ. ***y****es*).

В надписях вида ^X, ^G и других «шляпка» ^ обозначает кнопку Ctrl. То есть ^X — это то же самое, что и Ctrl+X.

Затем редактор предложит изменить имя файла, но делать этого не нужно — просто нажмите Enter.

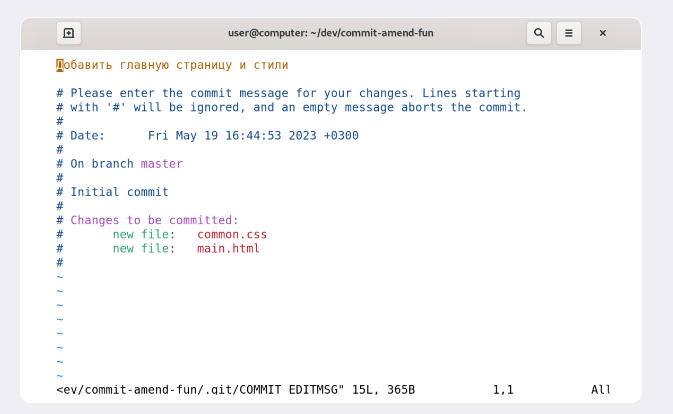


После нажатия Enter редактор закроется, а Git изменит сообщение последнего коммита.

Если вы не хотите менять сообщение через редактор, можно выйти из него с помощью Ctrl+X, а затем выбрать N (от англ. ***n****o*). В таком случае редактор закроется, и Git оставит сообщение последнего коммита «как было».

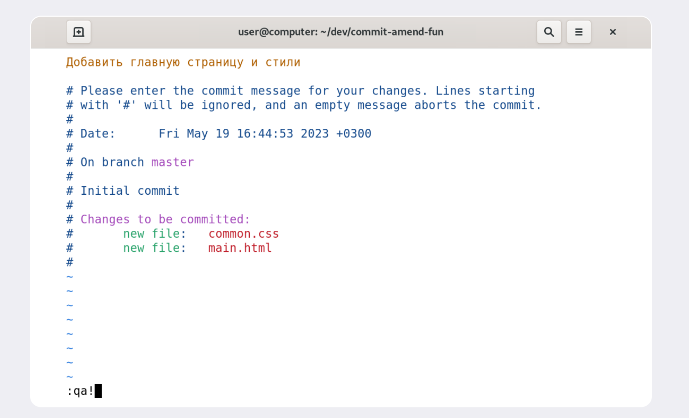
Vim — великий и ужасный

Если не откроется nano, то, скорее всего, откроется Vim.



Если вы ещё не умеете пользоваться Vim, мы рекомендуем сразу выйти из редактора и использовать флаг -m для указания сообщений коммита. Вот как выйти из Vim:

* Нажмите клавишу Esc.
* Наберите последовательность символов :qa!.
* Нажмите Enter.



После нажатия Enter редактор должен закрыться, а вы сможете продолжить работу так, как будто ничего и не открывалось.

**Vimtutor**

Редактор Vim — полезный инструмент, но его интерфейс сложно назвать интуитивным. И всё же во многих дистрибутивах это текстовый редактор по умолчанию. Если ваша текущая или будущая профессия связана с работой в консоли Linux, рекомендуем заглянуть в учебник по работе с Vim.

Для запуска учебника на русском языке достаточно выполнить команду vimtutor ru. Без ru запустится английская версия.

# **Как откатиться назад, если «всё сломалось»**

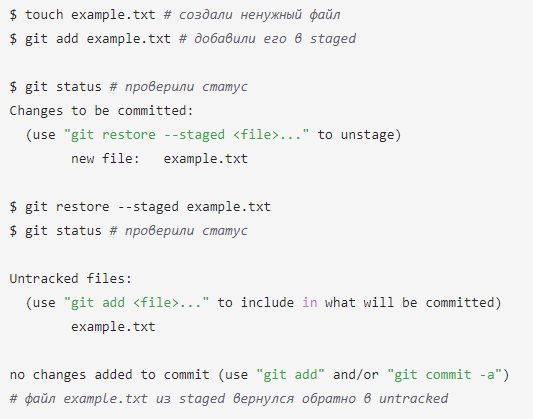
На разных этапах работы с Git могут происходить похожие ситуации:

* В список на коммит попал лишний файл (например, временный). Нужно «вынуть» его из списка.
* Последние несколько коммитов ошибочные: например, сделали не то, что было нужно, или нарушили логику. Хочется «откатить» сразу несколько коммитов, вернуть «как было вчера».
* Случайно изменился файл, который вообще не должен был меняться. Например, вы открыли не тот файл в редакторе и начали его исправлять.

### **Выполнить unstage изменений — git restore --staged <file>**

Допустим, вы создали или изменили какой-то файл и добавили его в список «на коммит» (staging area) с помощью git add, но потом передумали включать его туда. Убрать файл из staging поможет команда git restore --staged <file> (от англ. *restore* — «восстановить»).

В терминале это будет выглядеть примерно так.

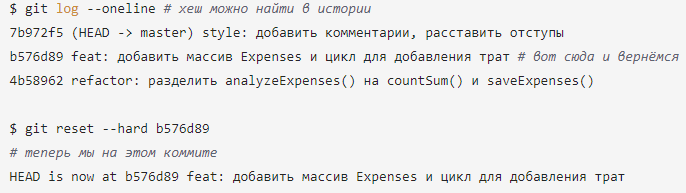


Вызов git restore --staged example.txt перевёл example.txt из staged обратно в untracked.

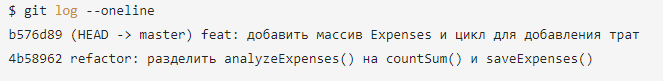
Чтобы «сбросить» все файлы из staged обратно в untracked/modified, можно воспользоваться командой git restore --staged .: она сбросит всю текущую папку (.).

### **«Откатить» коммит — git reset --hard <commit hash>**

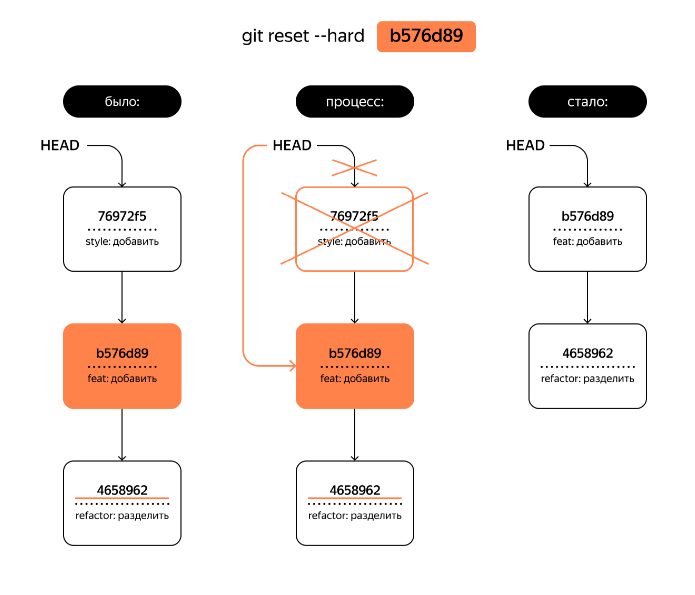
Иногда нужно «откатить» то, что уже было закоммичено, то есть вернуть состояние репозитория к более раннему. Для этого используют команду git reset --hard <commit hash> (от англ. *reset* — «сброс», «обнуление» и *hard* — «суровый»).



Теперь коммит b576d89 стал последним: вся дальнейшая разработка будет вестись от него. Файл также вернулся к тому состоянию, в котором был в момент этого коммита. А коммит 7b972f5 Git просто удалил. Это можно проверить, снова запросив лог. Он покажет следующее.



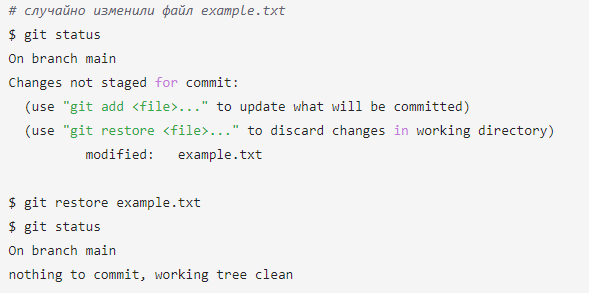
Вот так схематично выглядит весь процесс «отката» с помощью git reset --hard <hash>.



Будьте осторожны с командой git reset --hard! При удалении коммитов можно потерять что-то нужное.

### **«Откатить» изменения, которые не попали ни в staging, ни в коммит, — git restore <file>**

Может быть так, что вы случайно изменили файл, который не планировали. Теперь он отображается в Changes not staged for commit (modified). Чтобы вернуть всё «как было», можно выполнить команду git restore <file>.



Изменения в файле «откатятся» до последней версии, которая была сохранена через git commit или git add.

# **Просматриваем изменения в файлах**

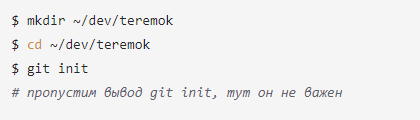
При работе с Git часто нужно узнать, что конкретно изменится или уже изменилось после того или иного коммита. Вот примеры таких ситуаций:

* Вы собираетесь сделать коммит, но хотите проверить (или перепроверить), какие именно изменения в него попадут.
* Вчера ваш коллега сделал коммит с сообщением small fix (англ. «небольшое исправление»), после чего тесты проекта начали «падать». Чтобы разобраться в ситуации, нужно посмотреть, что изменилось в этом коммите.

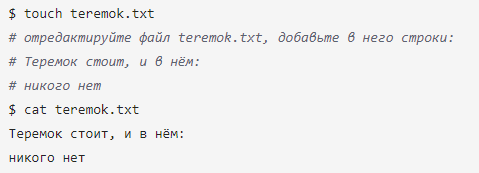
Всё это позволяет делать команда git diff (от англ. ***diff****erence* — «отличие», «разница»). О её возможностях пойдёт речь в этом и следующем уроках.

### **Подготавливаем репозиторий**

Историю масштабных и неожиданных изменений как нельзя лучше иллюстрирует сказка «Теремок». Создайте вместе с нами следующий репозиторий.



Добавьте файл teremok.txt и запишите в нём состояние теремка.

\

Теперь выполните коммит.

### **Селим мышку-норушку**

Откройте и отредактируйте файл teremok.txt, чтобы вместо никого нет стало Мышка-норушка. Сохраните файл, но не делайте коммит. Затем воспользуйтесь командой git status, чтобы посмотреть, что происходит с файлами.



Видно, что в теремке произошли какие-то изменения, но не видно, какие именно. Запустите git diff, чтобы выяснить детали. Эта команда сравнит последнюю закоммиченную версию файла teremok.txt с текущей (изменённой) версией.



Самое важное git diff выводит в конце:

* красный цвет строки никого нет значит, что эта строка была удалена;
* зелёный цвет строки Мышка-норушка значит, что она была добавлена.

Не все консоли умеют выводить цвета, поэтому строки помечаются не только цветом, но и знаком - или +. Минус — это удалённые строки, плюс — это добавленные.

Коротко разберём остальные строки вывода команды:

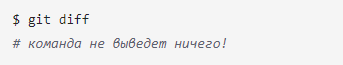
* Первые две строки (diff --git a/... b/... и index 901da07..ac459e1 100644) — это низкоуровневая техническая информация. Мы не будем на ней останавливаться.
* Строки --- a/teremok.txt и +++ b/teremok.txt говорят, что дальше будет выведен результат сравнения файлов a/teremok.txt и b/teremok.txt — исходной и текущей версий.
* Строка @@ -1,2 +1,2 @@ сообщает, какие строки файла попали в сравнение. Выражение 1,2 (неважно, с плюсом или с минусом) говорит, что были использованы две строки, начиная с первой. Если бы было, например, написано +15,7, это значило бы, что в сравнении участвуют *7*7 строк, начиная с *15*15-й.   
  Выражение со знаком минус (-1,2) относится к «оригинальной» версии файла (a/teremok.txt), а со знаком плюс (+1,2) — к «изменённой» (b/teremok.txt).

### **Просматриваем изменения в staging area**

Подготовьте мышку-норушку к коммиту, но пока не выполняйте его.

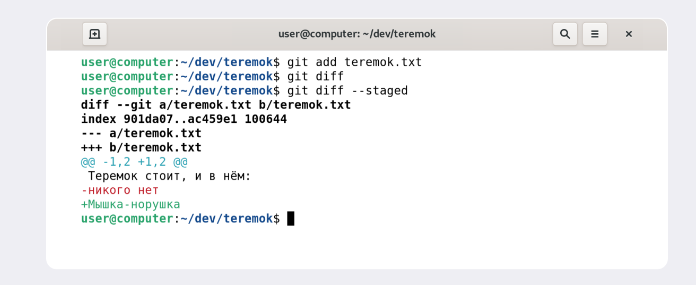


Вам наверняка знакома ситуация: вы выключили плиту и утюг, но вернулись проверить, всё ли точно в порядке. Сделайте что-то похожее с вашими изменениями: перепроверьте, что всё на месте.



Не волнуйтесь: изменения не потерялись. Просто по умолчанию команда git diff не показывает изменения в staged-файлах — только в modified.

Чтобы всё-таки просмотреть изменения в staged, нужно использовать флаг --staged: git diff --staged.



Все хорошо, изменения на месте! Было никого нет, а стало Мышка-норушка. Сделайте коммит.



# **Сопоставляем коммиты**

### **Дописываем строку в файл**

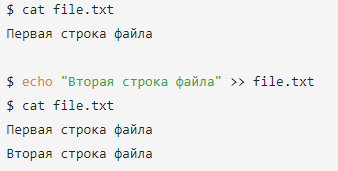
Чтобы продолжить сказку, вам нужно будет дописывать новые строки в конец файла teremok.txt. Для этого подходит команда echo (англ. «эхо»). Разберём её.

Сама по себе эта команда просто выводит в консоль то, что ей передали в качестве параметра.



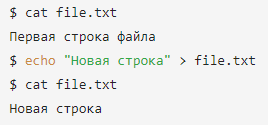
Но если скомбинировать echo с символами перенаправления вывода >> (два знака «больше»), то всё, что должно было попасть на экран, вместо этого будет записано в файл.

Скопировать кодBASH



Оператор >> — это возможность командной строки (Bash). Его можно использовать не только с echo, но и с любой другой командой, которая выводит что-то на экран.

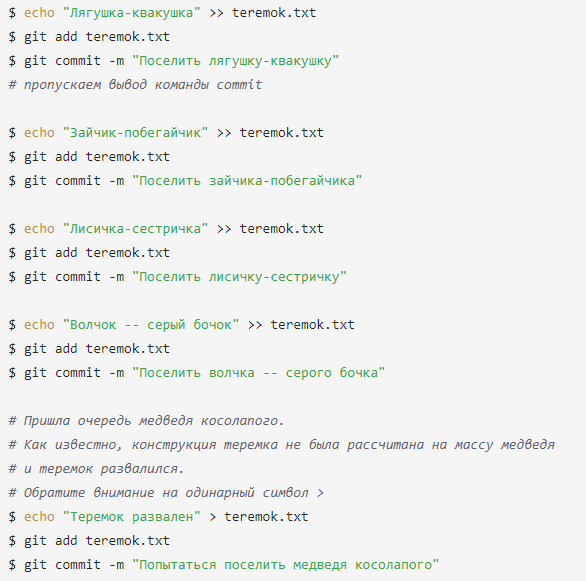
Одинарный символ > тоже перенаправит вывод команды в файл, но перед этим сотрёт содержимое файла, то есть перезапишет файл целиком.



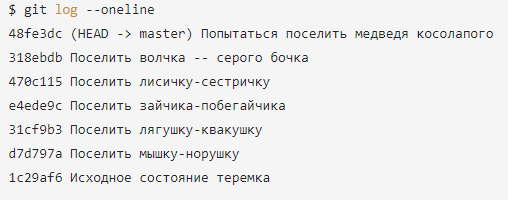
Обратите внимание, что после использования символа > Первая строка файла исчезла.

### **Селим всех остальных**

Вернёмся к сказке. Сделайте по коммиту для каждого нового персонажа. Используйте команду echo или дописывайте строки вручную в любом текстовом редакторе.



Если сейчас выполнить команду git log --oneline, получится примерно следующее.



Обратите внимание: у вас могут быть другие хеши, потому что они зависят в том числе от автора и времени, когда был сделан коммит.

### **Сравниваем коммиты**

Теперь, когда сказка рассказана и у вас есть история коммитов, попробуйте сравнить состояния файла teremok.txt между коммитами.

Например, чтобы получить максимально сокращённую версию сказки, можно сравнить состояние файлов на момент первого коммита (у нас это 1c29af6) и последнего (у нас 48fe3dc). Ваши хеши могут отличаться.

Передайте команде git diff хеши обоих коммитов. Состояние файлов на момент первого переданного коммита будет сравниваться с состоянием файлов на момент второго.



Вместо 48fe3dc можно было использовать HEAD: git diff 1c29af6 HEAD, потому что HEAD указывает на последний коммит.

Если коротко, это сказка о том, как теремок был и его не стало. А теперь попробуйте узнать, кто подселялся между лягушкой-квакушкой и волчком — серым бочком. Используйте соответствующие хеши коммитов.



Подселились: зайчик-побегайчик, лисичка-сестричка и волчок — серый бочок. Чёрные строки без знаков + или - — это «контекст», его git diff выводит только для того, чтобы стало понятно, что находится рядом с изменёнными (зелёными и красными) строками.

### **Порядок аргументов git diff**

По сути команда git diff A B выводит список инструкций: как превратить состояние A в состояние B. Если поменять A и B местами (git diff B A), то и инструкции будут обратные: как превратить B в A. При этом все зелёные строки станут красными, и наоборот.

Попробуйте git diff <конец сказки> <начало>. Вместо HEAD (конец сказки) можно также передать хеш.



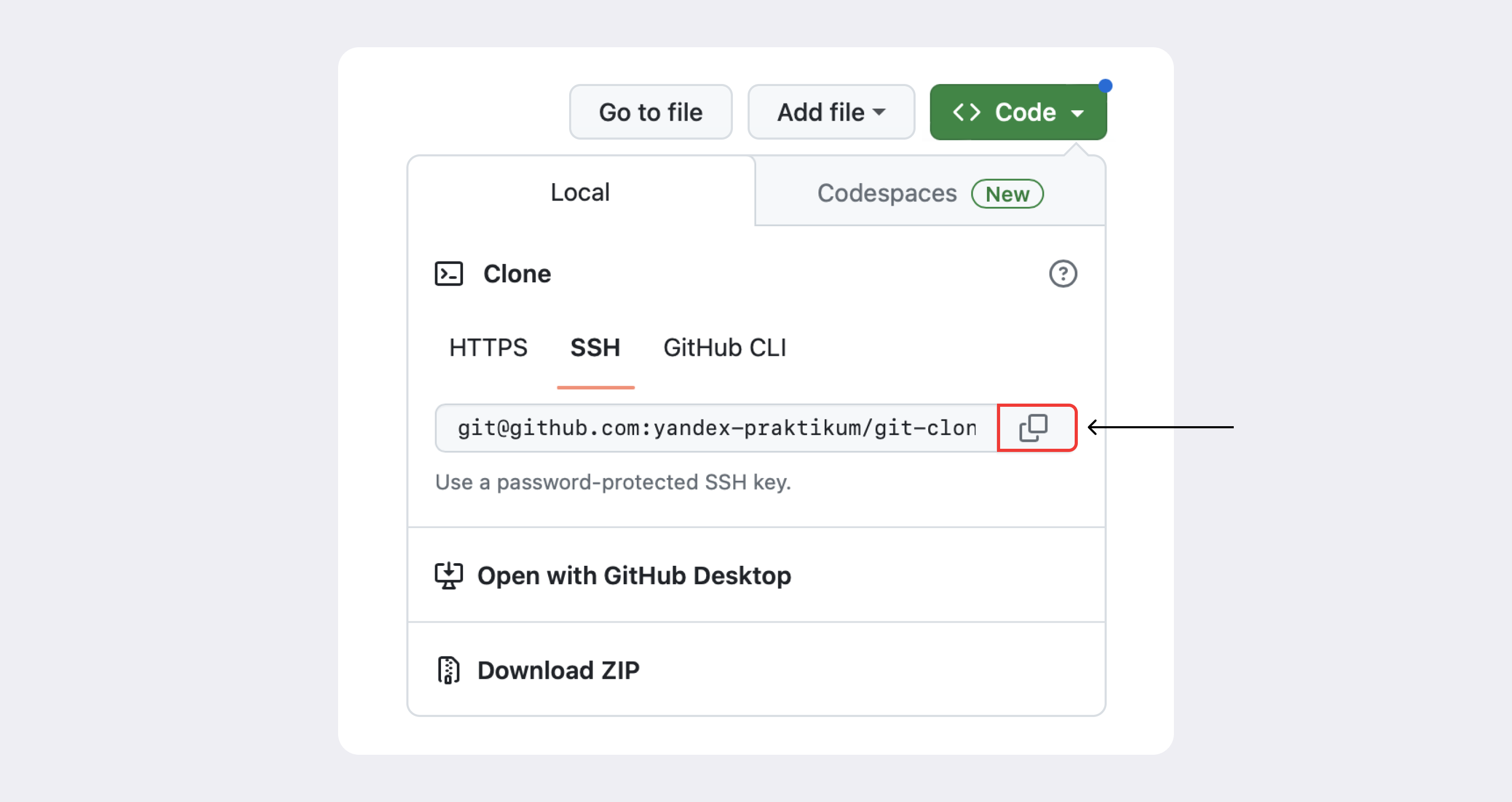
# **Клонируем репозиторий**

На платформе GitHub хранится множество проектов. Часть из них находится в открытом доступе. Такие open source проекты можно скачать на свой компьютер, внести какие-то правки или просто разобраться, как всё устроено внутри.

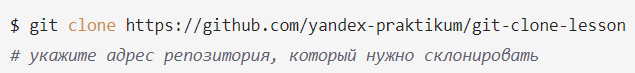
Процесс копирования удалённого репозитория на локальный компьютер называется **клонированием**. Клонирование репозитория — обычно первое, что делает разработчик на новом месте работы. Вы сымитируете эту ситуацию в уроке.

### **Клонировать репозиторий — git clone**

Откройте [этот репозиторий](https://github.com/yandex-praktikum/git-clone-lesson). Нажмите на зелёную кнопку Code. Появится окно со ссылкой. Если вы уже настроили SSH-ключ, убедитесь что выбрана опция SSH и нажмите на кнопку с двумя квадратами справа — она скопирует ссылку в буфер обмена. Вы также можете скопировать ссылку вручную.

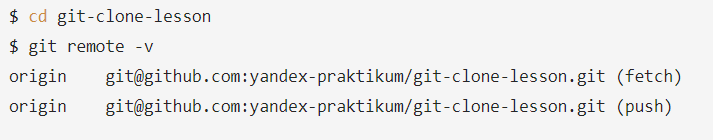


Теперь откройте консоль, перейдите в папку, в которую хотите положить репозиторий, и выполните команду git clone (от англ. *clone* — «клон», «копия»). Она создаст копию удалённого репозитория на вашем компьютере. В качестве параметра команде нужно передать адрес репозитория, который вы только что скопировали на GitHub.



Команда git clone автоматически связывает локальный и удалённый репозиторий. То есть если в GitHub-репозитории что-то поменяется (например, добавятся коммиты), вам не нужно будет заново клонировать его. Достаточно будет выполнить команду, которая обновит вашу копию.

Убедитесь в том, что репозитории связаны, командой git remote -v.



# **Выполняем Fork**

Допустим, вы хотите усовершенствовать чужой проект или как-то использовать его в своей работе, но у вас нет прав на изменение оригинального репозитория.

В этом уроке разберём ещё одну полезную операцию копирования проектов. В отличие от клонирования, она не скачает репозиторий на локальный компьютер, но добавит его прямо в ваш аккаунт на сервере GitHub.

### **Что такое Fork**

**Fork** (англ. «развилка», «ответвление»), или **«форк»**, — это GitHub-операция; напрямую с Git она не связана. «Форк» создаёт копию репозитория в аккаунте GitHub. Такая копия будет полностью независима. Изменения, которые вы внесёте, не будут синхронизированы с исходным репозиторием.

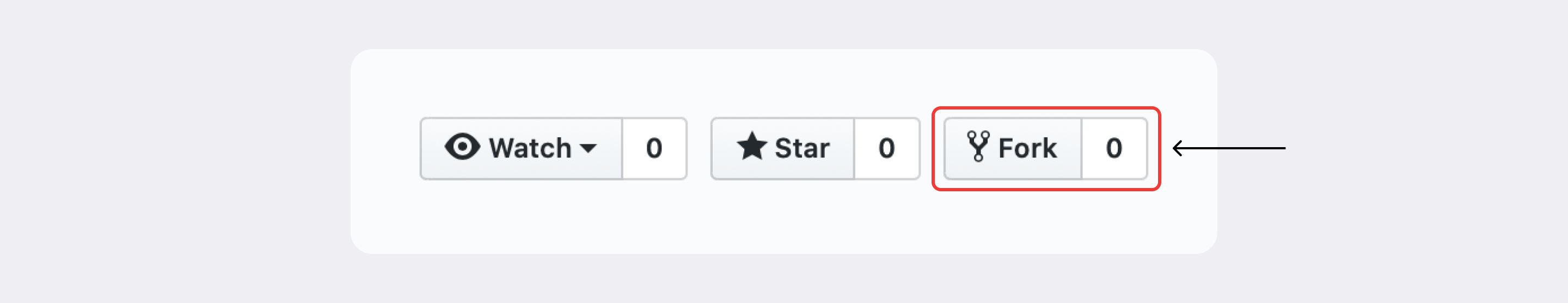
В процессе «форка» создаётся копия всех файлов, истории коммитов и веток. Эта копия сохраняется в вашей учётной записи GitHub.

Вот некоторые из распространённых причин использования «форков»:

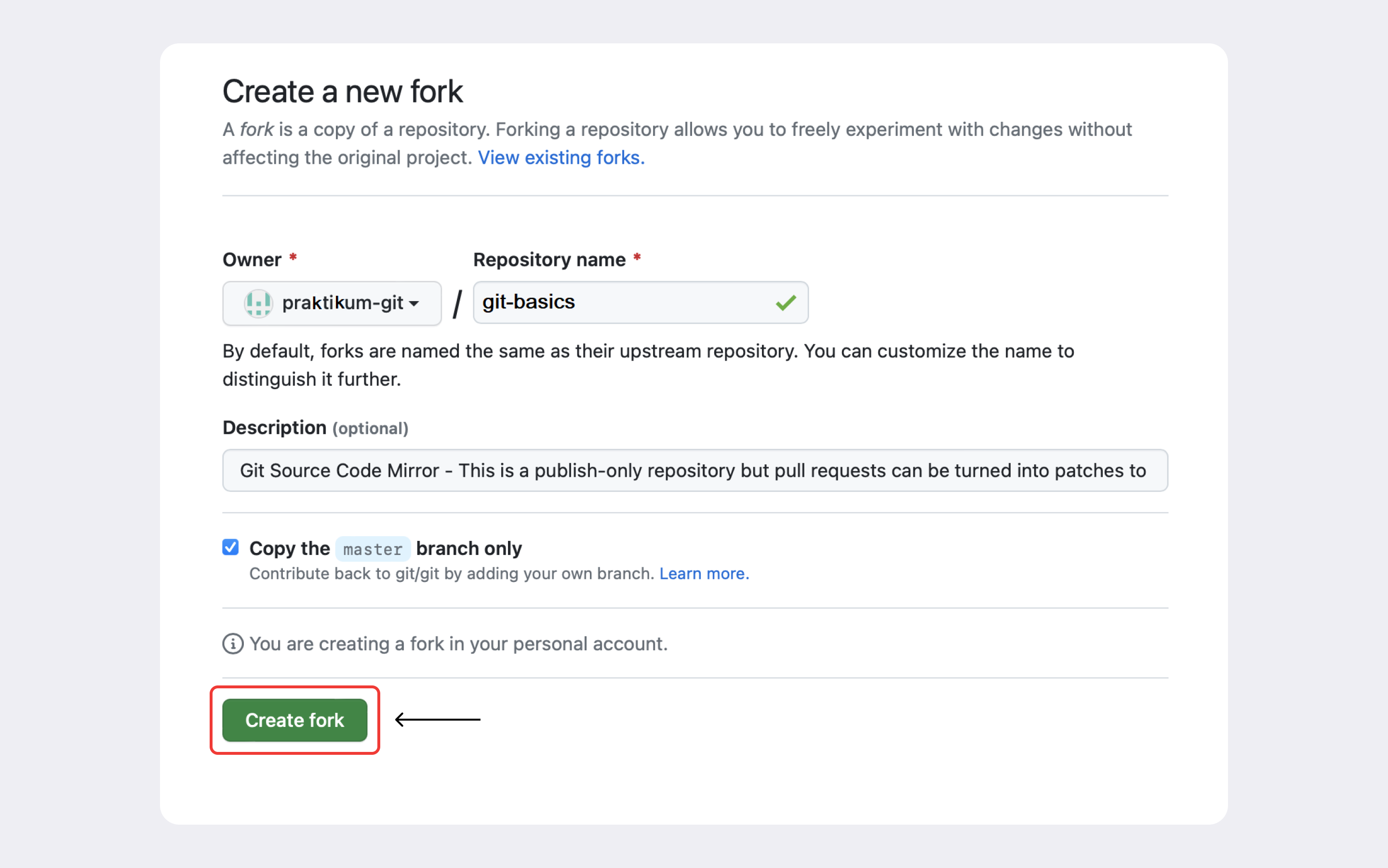
* Вы хотите внести свой вклад в проект (например, open source), но не имеете прав на изменение исходного репозитория. Тогда вы можете сделать «форк», добавить нужные правки, а затем отправить запрос на включение этих изменений в оригинальный проект.
* Вы хотите развивать проект независимо от исходного. Допустим, создатели проекта решили, что не будут добавлять функциональность, которая вам необходима. В таком случае вы можете сделать «форк» и добавить её самостоятельно.

### **Применяем Fork**

Потренируйтесь выполнять «форк». Перейдите [по этой ссылке](https://github.com/yandex-praktikum/git-basics) и нажмите на кнопку Fork в правом верхнем углу.



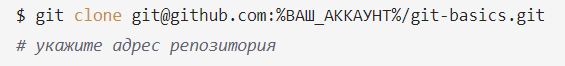
В открывшемся окне вы можете поменять название и описание репозитория. Или поставить галку, чтобы склонировать только главную ветку вместо всех сразу. Нажмите **Create fork** (англ. *\*\**«создать копию репозитория»).



Немного подождите, пока репозиторий скопируется. После этого он будет доступен по адресу <https://github.com/%USERNAME%/git-basics>, где %USERNAME% — ваше имя пользователя.

В результате вы получите полную копию исходного репозитория, которую можно свободно изменять и которой можно управлять.

Теперь внесите изменения в новый репозиторий. Для начала скопируйте его локально.



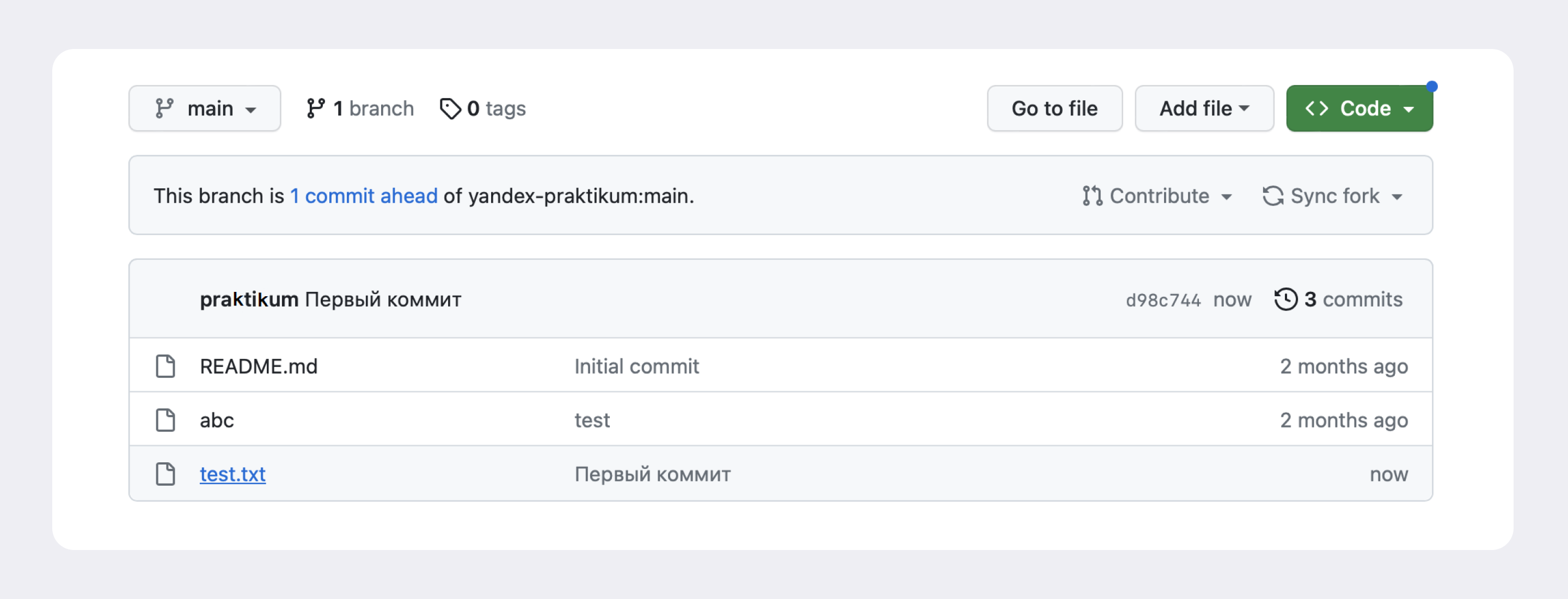
Когда клонирование будет завершено, перейдите в локальный репозиторий командой cd.

Создайте файл test.txt, добавьте в него несколько любых предложений и выполните коммит git commit -m "Первый коммит".

Если ввести git commit без флага -m, откроется редактор Vim. Чтобы выйти из него, нажмите клавишу Esc, наберите последовательность символов :q! и нажмите Enter.

Отправьте изменения в удалённый репозиторий командой git push (от англ. *push* — «толкнуть», «протолкнуть»). Обратите внимание: вам не нужно указывать флаг -u origin main. Команда git clone сама свяжет удалённые и локальные ветки в момент клонирования.

Снова откройте браузер с GitHub и обновите страницу. Вы увидите опубликованные вами файл и коммит.



## **Задание для самостоятельной работы**

Проект необязательно начинается с нуля: иногда вдохновение можно найти на просторах GitHub, но при этом решить задачу по-своему.

Например, на GitHub есть много репозиториев в формате **awesome lists** (англ. «потрясающие списки»). Это репозитории, в которых собраны разные полезные ссылки со всего интернета. Вот некоторые из них:

* [Шрифты](https://github.com/brabadu/awesome-fonts#readme)
* [Публичные выступления](https://github.com/matteofigus/awesome-speaking#readme)
* [Удалённая работа](https://github.com/lukasz-madon/awesome-remote-job#readme)

Выберите один из этих проектов или любой другой понравившийся. Выполните его «форк», а затем склонируйте локально.

Ура! Вы научились «форкать» репозитории. Вот о чём важно помнить:

* «Форк» позволяет получить точную копию GitHub-репозитория в ваш аккаунт.
* Копия, которая получена с помощью «форка», полностью независима от оригинального проекта — изменения не будут синхронизированы.