**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.

💡 Важно: опция --amend работает только с последним коммитом (HEAD). Для исправления более ранних коммитов есть другие команды. Мы рассмотрим их в следующих модулях курса.

**Подготавливаем репозиторий**

Создайте тренировочный репозиторий для отработки команды.

Скопировать код

BASH

$ mkdir ~/dev/commit-amend-fun

$ cd ~/dev/commit-amend-fun

$ git init

# пропустим вывод git init

**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.

Скопировать код

BASH

$ touch main.html

$ touch common.css

# дальше отредактировали оба файла

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

Скопировать код

BASH

$ git add main.html

$ git commit -m "Добавить главную страницу"

$ git log --oneline

777fec3 Добавить главную страницу

Напомним, что если вы повторяете за нами эти действия локально, у вас будут другие хеши коммитов.

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

Скопировать код

BASH

$ git status

On branch main

Untracked files:

(use "git add <file>..." to include in what will be committed)

common.css

nothing added to commit but untracked files present (use "git add" to track)

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

Скопировать код

BASH

$ git add common.css

# добавили файл common.css в список на коммит как обычно

# но вместо команды commit -m '...'

# будет:

$ git commit --amend --no-edit

$ git log --oneline

8340eb2 Добавить главную страницу

# коммит в истории всё ещё один (но у него новый хеш)

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.

Скопировать код

BASH

# ещё раз отредактировали main.html

$ git add main.html # добавили в список на коммит

$ git commit --amend --no-edit

💡 **Что же выбрать: новый коммит или --amend?**

В нашем примере вместо изменения последнего коммита можно было также выполнить новый коммит с одним файлом common.css. Кажется, что так проще, но добавить изменения в уже существующий коммит может быть правильнее.

Например, через месяц кто-то захочет просмотреть историю изменений. Намного проще понять, что изменилось, если оба файла находятся в одном коммите. Иначе коммит со второй порцией изменений придётся искать.

**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.

Скопировать код

BASH

$ git commit --amend -m "Добавить главную страницу и стили"

$ git log --oneline

a31fa24 Добавить главную страницу и стили

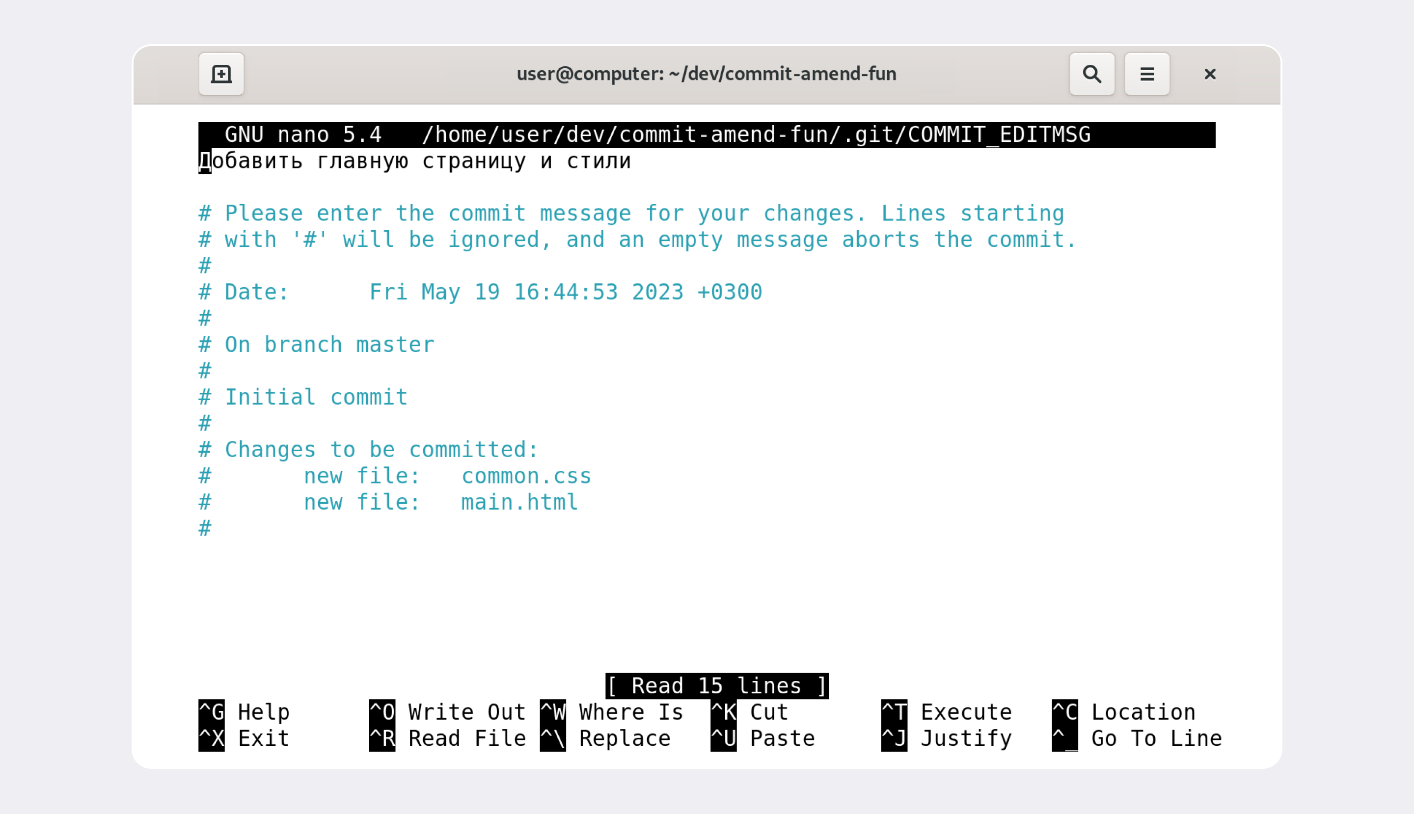
Хеш коммита снова поменялся, потому что изменились сообщение и время коммита. При этом файлы в коммите остались те же: main.html и common.css.

**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

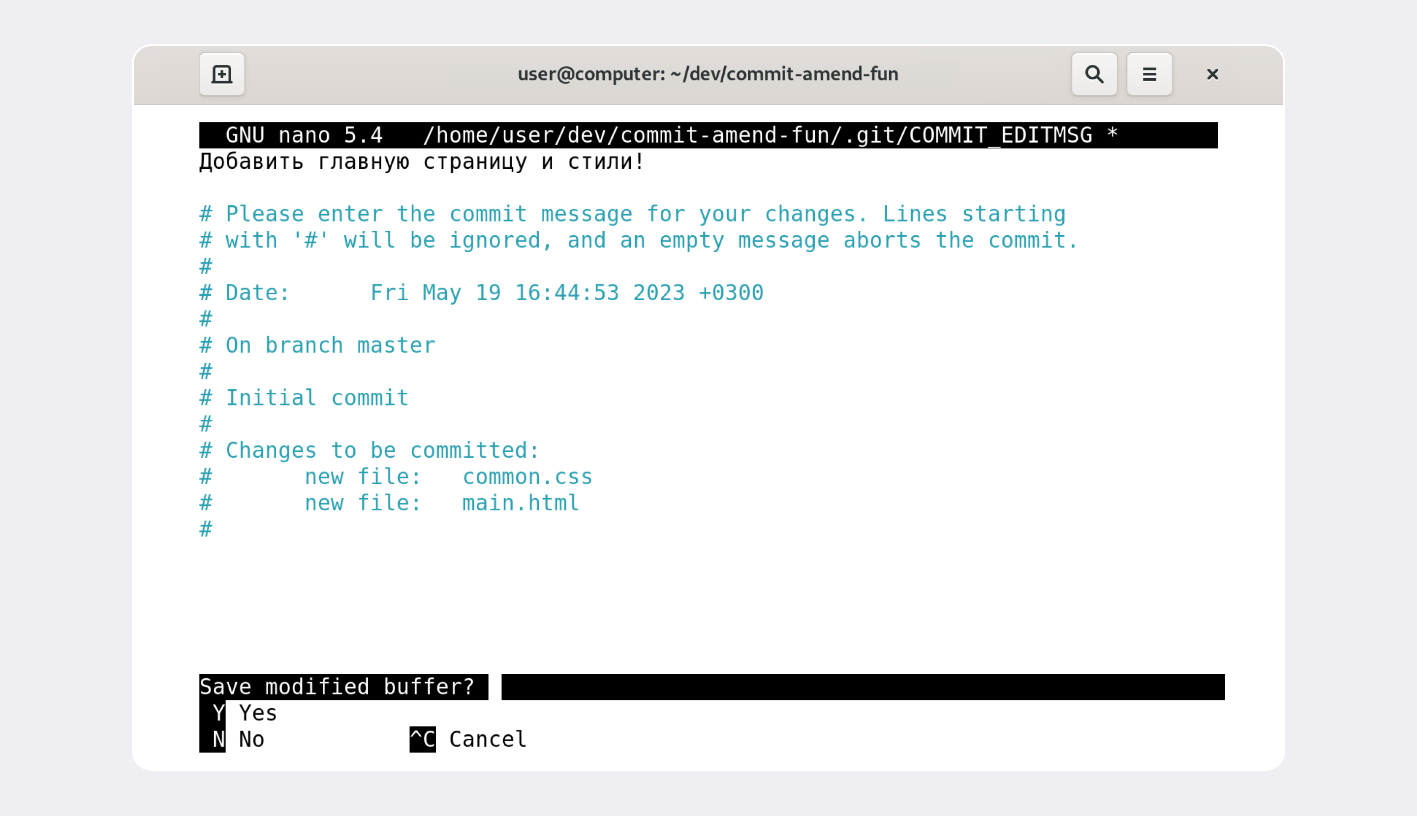
При редактировании сообщения коммита редактор GNU nano выглядит примерно так.



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

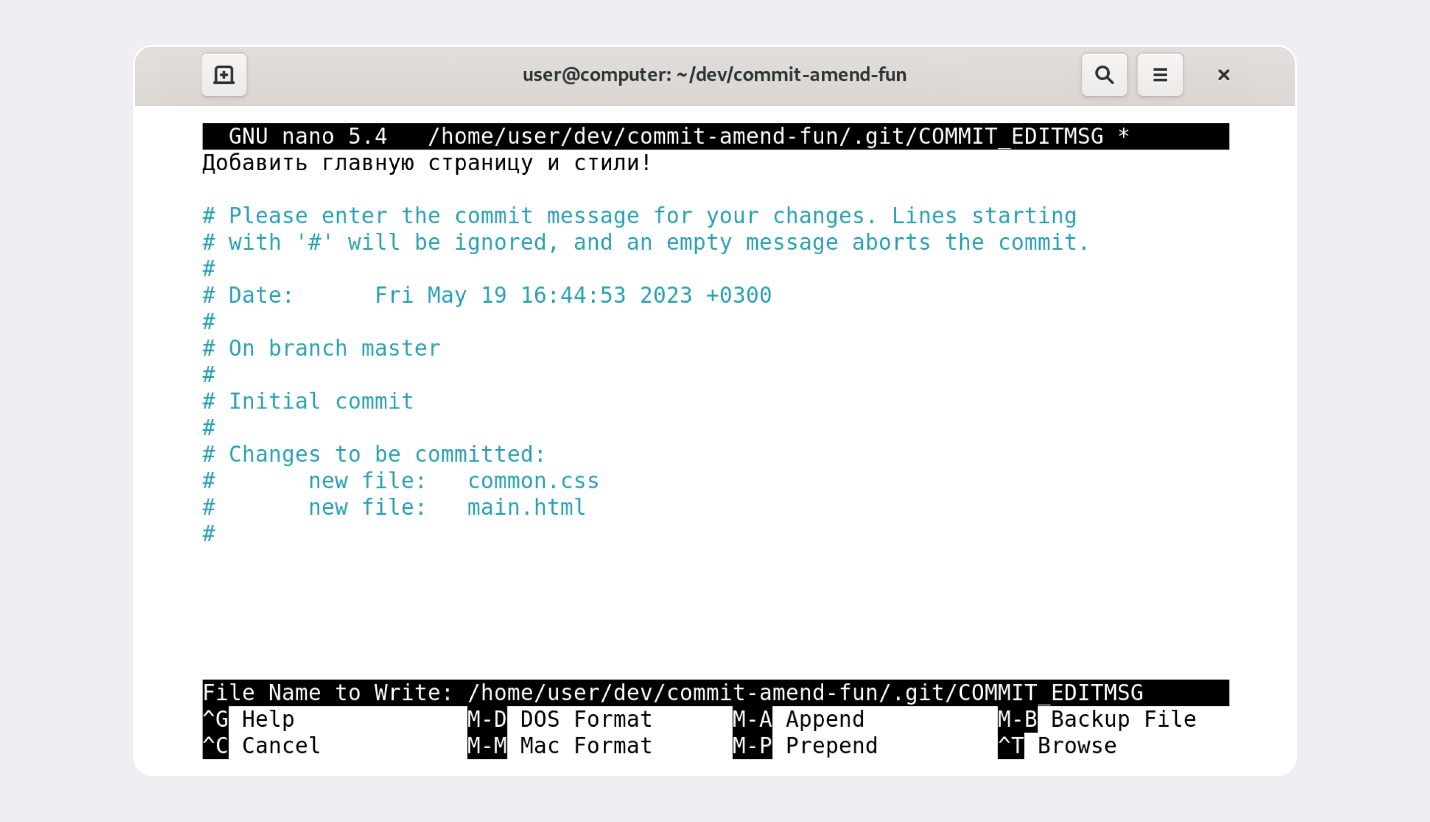
Допустим, решили добавить в конце сообщения восклицательный знак. Чтобы сохранить новое сообщение, нужно нажать Ctrl+X, где X значит *e****x****it* (англ. «выход»).

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



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

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

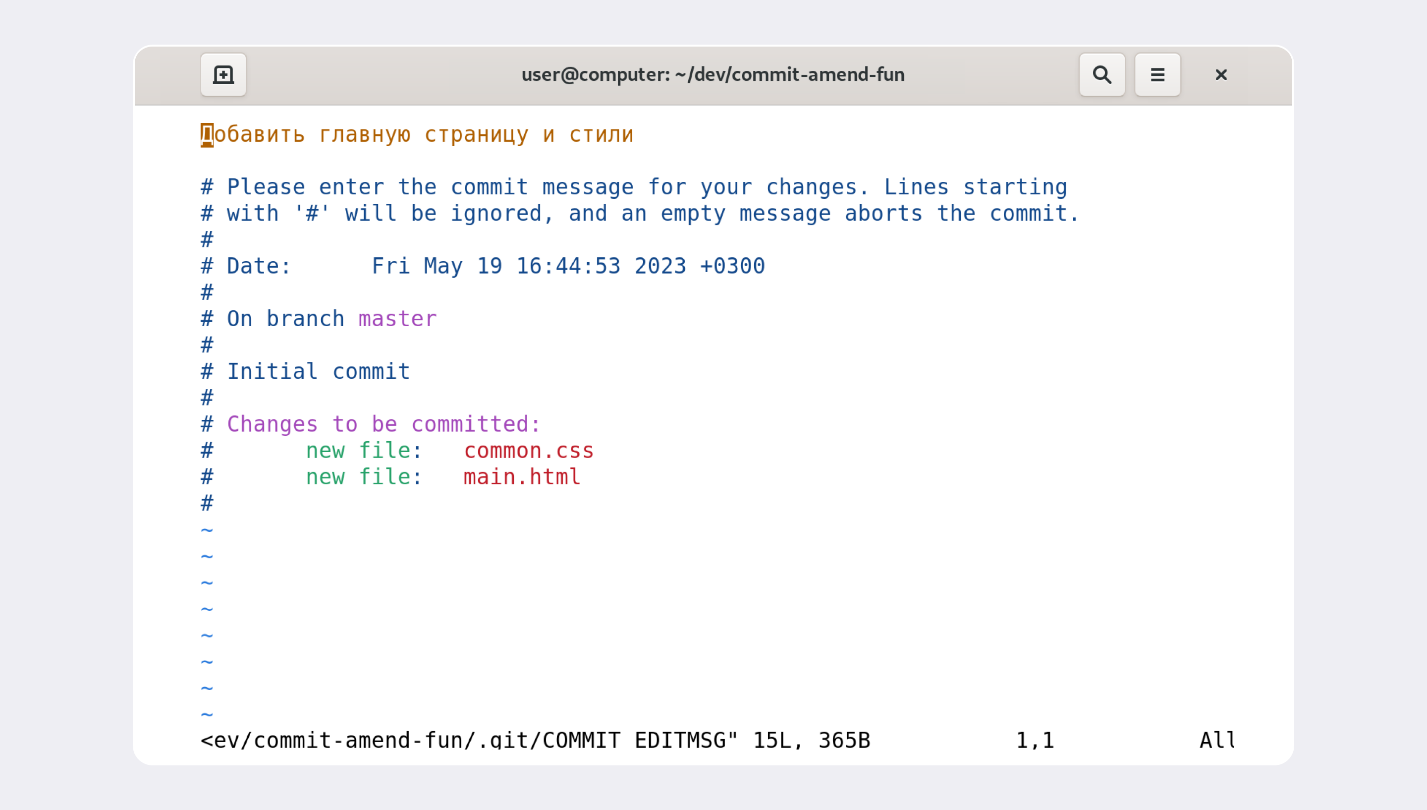


После нажатия Enter редактор закроется, а Git изменит сообщение последнего коммита.

💡 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».

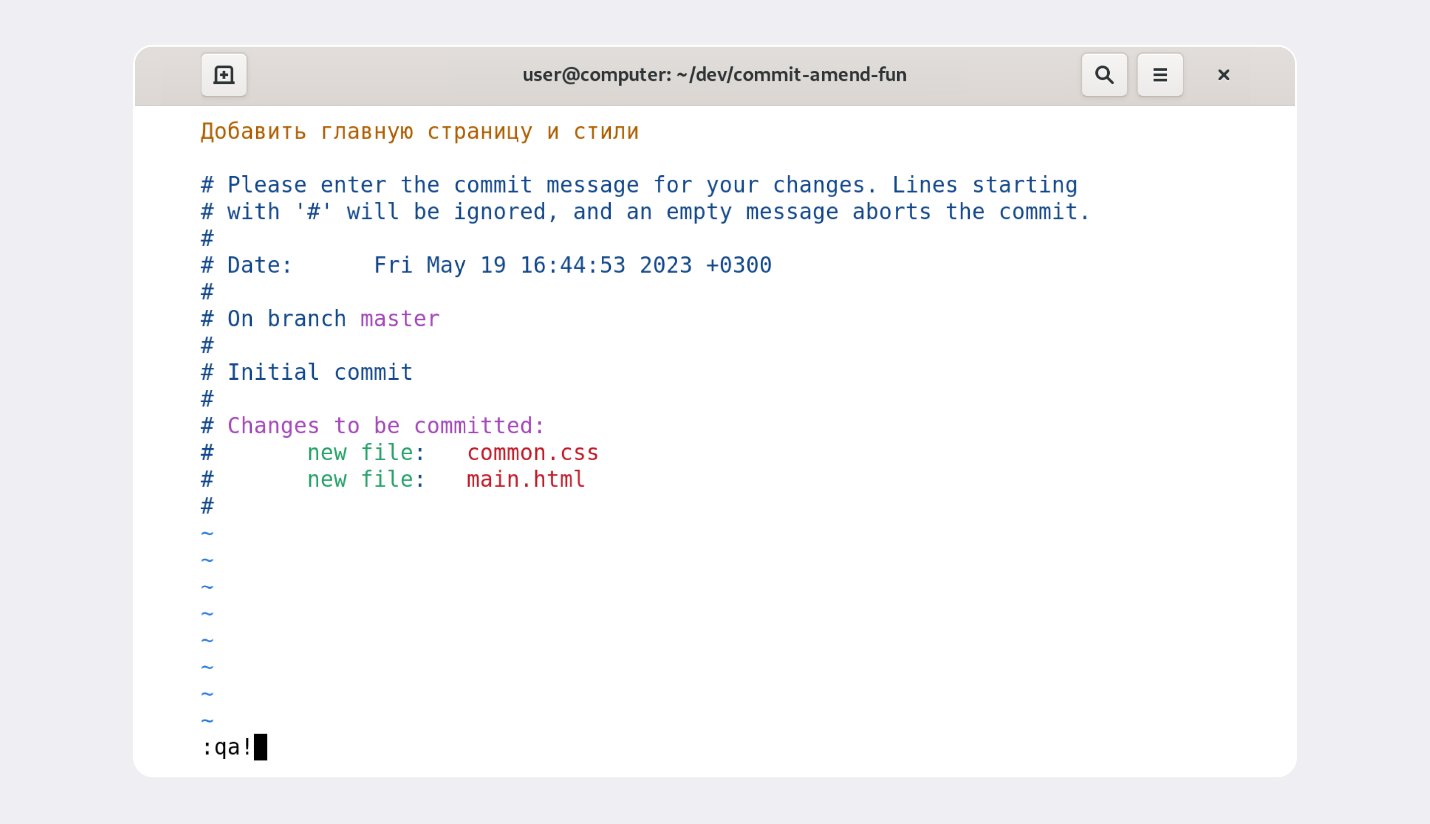
Vim — великий и ужасный

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:

1. Нажмите клавишу Esc.
2. Наберите последовательность символов :qa!.
3. 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.

Хорошо! В ходе работы вам наверняка ещё не раз понадобится внести какие-то исправления в историю коммитов. Флаг --amend — ваш союзник. Коротко подытожим урок:

--amend рассчитан на работу с последним коммитом (HEAD).

Дополнить коммит новыми файлами можно с помощью git commit --amend --no-edit. Благодаря опции --no-edit сообщение к коммиту останется таким, каким и было.

Изменить сообщение к коммиту позволяет команда git commit --amend -m "Обновлённое сообщение коммита".

# 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.

В этом уроке рассмотрим такие случаи и научим вас «откатывать» нежелательные изменения.

### 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 выводе команды git status есть подсказка в скобках: use "git restore --staged <file>..." to unstage. Так что, даже если вы и забыли эту команду, Git напомнит вам.

В терминале это будет выглядеть примерно так.

Скопировать код

BASH

$ touch example.txt # создали ненужный файл

$ git add example.txt # добавили его в staged

$ git status # проверили статус

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

new file: example.txt

$ git restore --staged example.txt

$ git status # проверили статус

Untracked files:

(use "git add <file>..." to include in what will be committed)

example.txt

no changes added to commit (use "git add" and/or "git commit -a")

# файл example.txt из staged вернулся обратно в untracked

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 (.).

💡 Раньше для этой операции использовали команду git reset HEAD. Но git reset «перегруженная»: она умеет делать много разных действий в зависимости от переданных параметров. Чтобы было меньше путаницы, в Git была добавлена отдельная команда git restore.

### «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»).

Скопировать код

BASH

$ git log --oneline # хеш можно найти в истории

7b972f5 (HEAD -> master) style: добавить комментарии, расставить отступы

b576d89 feat: добавить массив Expenses и цикл для добавления трат # вот сюда и вернёмся

4b58962 refactor: разделить analyzeExpenses() на countSum() и saveExpenses()

$ git reset --hard b576d89

# теперь мы на этом коммите

HEAD is now at b576d89 feat: добавить массив Expenses и цикл для добавления трат

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.

Скопировать код

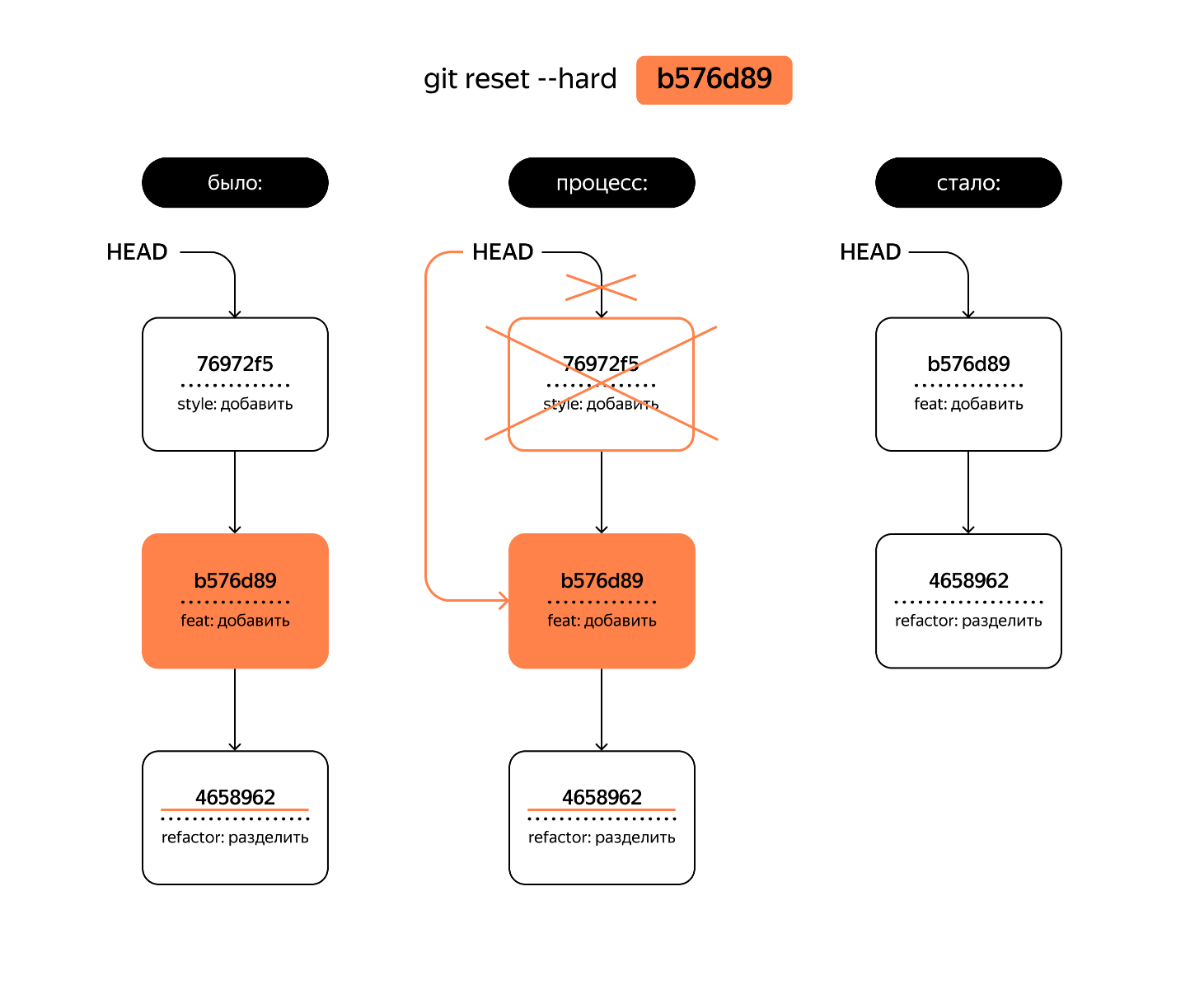
BASH

$ git log --oneline

b576d89 (HEAD -> master) feat: добавить массив Expenses и цикл для добавления трат

4b58962 refactor: разделить analyzeExpenses() на countSum() и saveExpenses()

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>.

Скопировать код

BASH

# случайно изменили файл example.txt

$ git status

On branch main

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: example.txt

$ git restore example.txt

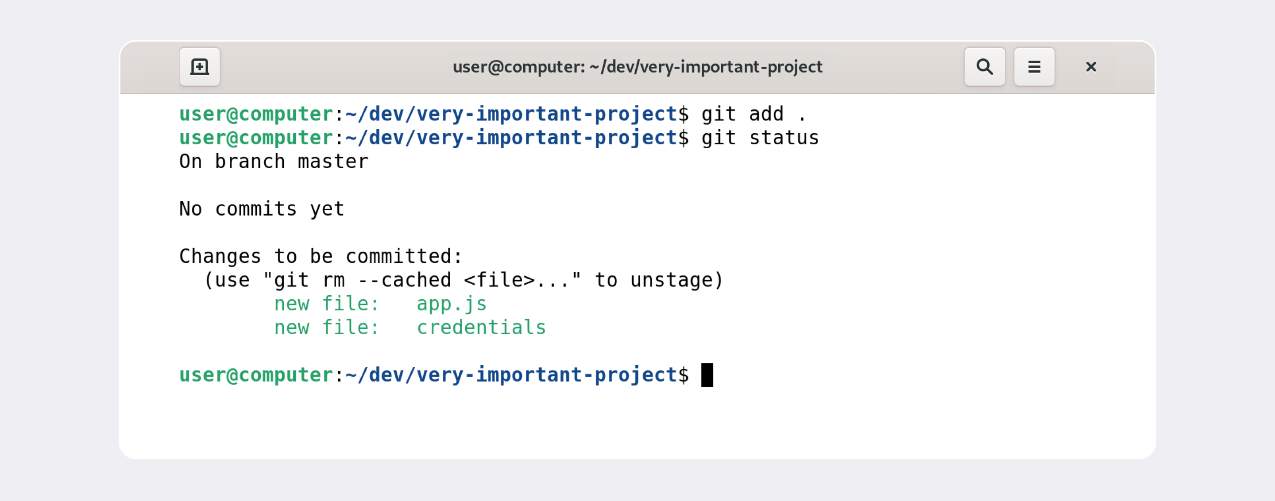
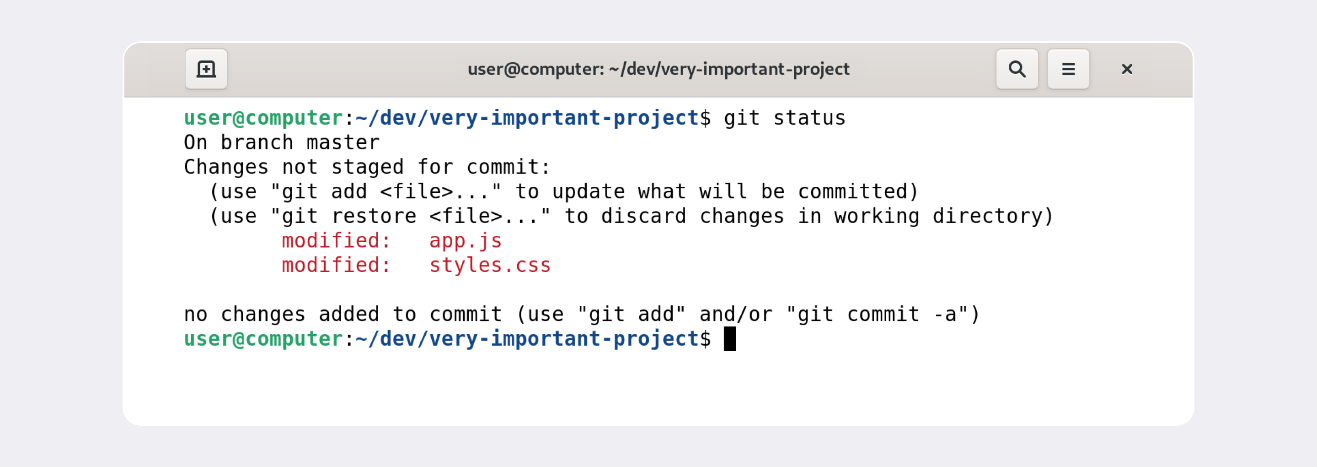
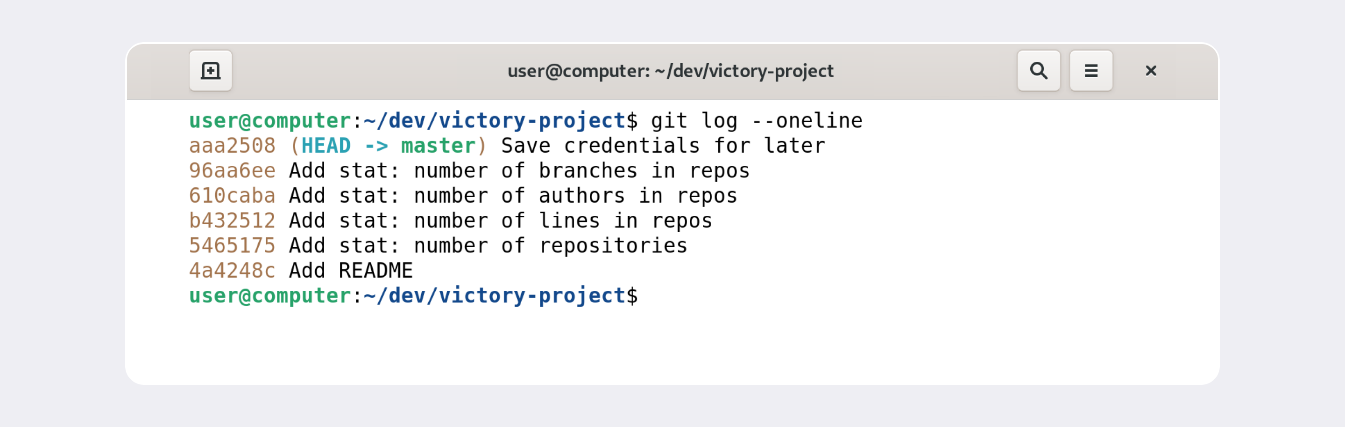
$ git status

On branch main

nothing to commit, working tree clean

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

Ура! Теперь вы знаете, как сделать шаг назад, если что-то пошло не так. Вот о чём мы рассказали:

* Команда git restore --staged <file> переведёт файл из staged обратно в modified или untracked.
* Команда git reset --hard <commit hash> «откатит» историю до коммита с хешем <hash>. Более поздние коммиты потеряются!
* Команда git restore <file> «откатит» изменения в файле до последней сохранённой (в коммите или в staging) версии.
* **Практическая работа №3. Нужно откатить изменения**
* Говорят, что ошибаться — это нормально и даже хорошо. Каждая ошибка учит чему-то новому. Но ещё полезнее учиться на чужих ошибках.
* В этой практической работе вы окунётесь в мир программирования вместе с нашим героем Геной: узнаете, какие опасности его подстерегали, и исправите типичные ошибки новичка с помощью инструментов «отката» Git. Поехали!
* **Секреты на сцене**
* В понедельник программист Гена получил задание: ему поручили написать небольшой сервис. Сервис должен подключаться к базе данных в AWS и генерировать её описание. Проект было решено опубликовать на GitHub с открытым исходным кодом, чтобы упрочить репутацию компании.
* Петя, наставник Гены, прислал ему файл credentials и объяснил: файл содержит ключи, которые позволят приложению получить доступ к тестовому аккаунту AWS — в том числе к тестовым базам.
* Гена приступил к работе. Он не знал, что файл должен быть в папке ~/.aws, и сохранил его в папку с проектом: ~/dev/very-important-project/credentials. Позже он прочитал документацию и скопировал файл в нужную папку, но из папки с проектом удалить забыл.
* Наш герой не любит лишних действий. Чтобы не добавлять файлы в staging area по одному, он всегда использует команду git add ..
* 
* Когда Петя увидел Генин список файлов на коммит, то очень испугался: даже начал махать руками и восклицать что-то на программистском.
* Если коротко
* Начало формы
* Почему Петя был взволнован?
* 
* Не стоит коммитить по два файла сразу. Лучше делать так: один файл — один коммит. Особенно если это файлы разных типов: в app.js — код приложения, а в credentials — ключи.
* Коммитить можно сколько угодно файлов сразу. Также они могут быть разных типов — в этом нет ничего страшного.
* 
* Первый коммит в любом проекте должен содержать файл README.md. Именно так GitHub понимает, какой файл отображать в качестве приветственной страницы.
* Необязательно: файла README.md может и вовсе не быть в проекте.
* 
* Петя волновался, что Гена выполнит команду git commit без него и напишет какое-нибудь неинформативное сообщение.
* Сообщения коммитов важны, но не настолько, чтобы махать руками. К тому же сообщение легко исправить с помощью команды git commit --amend -m ....
* 
* Правильный ответ
* Файл credentials позволяет получить доступ к AWS-аккаунту, а проект будет с открытым кодом. Кто-нибудь может подсмотреть ключи и использовать их.
* Файлы с ключами и паролями вообще не стоит коммитить в Git-репозитории, особенно в публичные.
* Конец формы
* ⚠️ Проблема с файлом credentials реальная. Известны случаи, когда разработчики случайно коммитили и публиковали на GitHub ключи доступа к аккаунтам и системам. Иногда эти ключи находили и использовали в корыстных целях злоумышленники.
* Гена понял, в чём проблема: нужно убрать из списка на коммит файл credentials. Но он не знает, как её исправить, поэтому на помощь пришёл Петя.
* Введите в поле ниже команду в формате git ... (многоточие нужно заменить на конкретную команду), которую Петя выполнит в первую очередь. Обратите внимание: Петя хочет убрать только файл credentials и при этом оставить файл app.js в списке на коммит. Так Гене будет проще продолжить работу.
* Ваш ответ неправильный
* git remove credentials
* Мы ожидали, что вы введёте команду git restore --staged credentials. Эта команда уберёт файл credentials из списка на коммит, и он вернётся в состояние untracked. Дальше Гена может удалить этот файл, чтобы не мешал.
* **Когда написал ненужное**
* Приложение, над которым работает Гена, развивается. Пользователи требовательны: нужно ещё больше разных кнопок и диаграмм. В проекте даже появился файл styles.css, в котором записаны цвета и стили для всех типов элементов.
* Но однажды Гене надоело создавать кнопки. Он мечтал добавить в своё приложение светлую и тёмную темы. При этом всё должно работать автоматически: если у пользователя за окном темно, то тема должна быть тёмная. А если светло — светлая.
* Гена долго писал логику: определял по IP-адресу пользователя примерные географические координаты, потом вычислял время рассвета и заката в этой точке. Даже добавил проверку погоды: если в городе пользователя гроза, тема переключается на тёмную.
* Пришло время отправить код на ревью Пете. И вот каким был ответ наставника: «В этом коде нет необходимости: во всех браузерах уже есть логика автоматической смены темы со светлой на тёмную и наоборот. Достаточно добавить в styles.css пару строк».
* Гена расстроился, но не сильно, ведь он узнал столько нового, пока возился с этой задачей! Теперь ему нужно откатить изменения в файле app.js, но при этом оставить styles.css нетронутым: стили добавил Петя, а Гена пока не умеет с ними работать.
* Посмотрите на скриншот и выполните задание.
* 
* Введите в поле команду, которую должен выполнить Гена, чтобы откатить изменения в файле app.js, но при этом оставить файл styles.css как есть.
* Ваш ответ правильный
* git restore app.js
* Да, всё так! Команда git restore <file> выручит вас, если нужно откатить изменения, которые ещё не попали ни в staging area, ни в коммит.
* **Когда уже накоммитил**
* Гена проработал в компании ещё пять лет. Однажды он и сам стал наставником новичка Виктора.
* Виктору поручили небольшой проект, который собирает статистику по другим проектам. Всё повторяется: проект нужно делать с использованием AWS. Гена привычно отправил Вите файл credentials с ключами доступа к тестовому аккаунту.
* Витя бесстрашно взялся за проект, а в конце рабочего дня показал Гене, что успел сделать. Для этого он использовал команду git log --oneline.
* 
* Последний коммит aaa2508 с сообщением Save credentials for later что-то смутно напомнил Гене: наставник Петя (давно ушедший из компании), возгласы и взмахи руками, и чувство, что опасность совсем рядом.
* Гена решил проверить свои подозрения и посмотреть содержимое последнего коммита.
* 
* Витя закоммитил файл с ключами доступа к AWS-аккаунту! Теперь нужно срочно удалить этот коммит. К счастью, Витя ещё не успел опубликовать последние коммиты на GitHub. Гена поспешил на помощь.
* Какую команду нужно использовать Гене, чтобы коммит aaa2508 исчез? Введите её в поле ниже.
* Ваш ответ неправильный
* git reset --hard <aaa2508>
* Мы ожидали, что вы введёте команду git reset --hard 96aa6ee, которая откатит состояние репозитория к предпоследнему коммиту. Последний коммит (aaa2508) при этом исчезнет из истории.
* На этом рассказ окончен. Наш герой стал опытным разработчиком и учит новичков на своих ошибках. Теперь вы, как и Гена, мастер откатов! В следующий раз, когда что-то пойдёт не так, вы не растеряетесь. git reset --hard <time travel!>

# 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.

Скопировать код

BASH

$ mkdir ~/dev/teremok

$ cd ~/dev/teremok

$ git init

# пропустим вывод git init, тут он не важен

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

Скопировать код

BASH

$ touch teremok.txt

# отредактируйте файл teremok.txt, добавьте в него строки:

# Теремок стоит, и в нём:

# никого нет

$ cat teremok.txt

Теремок стоит, и в нём:

никого нет

Теперь выполните коммит.

Скопировать код

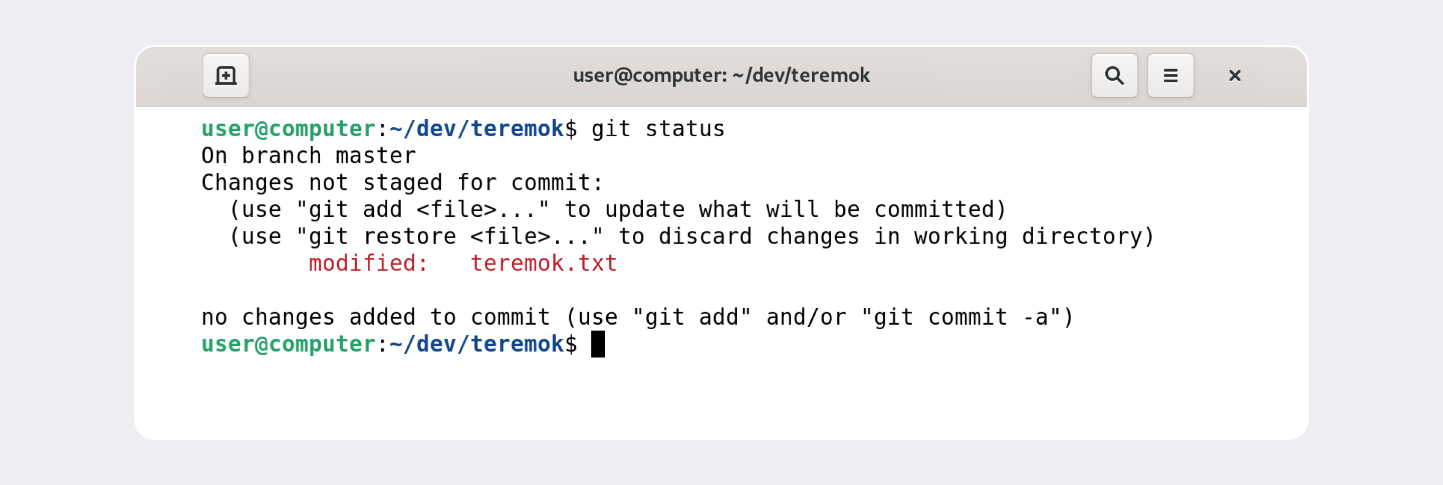
BASH

$ git add teremok.txt

$ git commit -m "Исходное состояние теремка"

### 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 77 строк, начиная с 1515-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).

💡 **Зачем вообще указывать, какие строки файла участвуют? Разве сравниваются не все строки?**

Указывается не то, какие строки сравнивались, а какие попали в вывод команды git diff. Это важно для больших файлов. Если, например, сравнить два файла по 10001000 строк, в которых отличается только 500500-я строка, то git diff выведет порядка 1010 строк (что-нибудь вроде @@ -495,10 +495,10 @@ — с 495495-й по 505505-ю). Иначе пришлось бы читать всю тысячу. 1010 строк вместо одной нужно, чтобы было проще понять контекст изменения.

### Просматриваем изменения в staging area

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

Скопировать код

BASH

$ git add teremok.txt

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.

Скопировать код

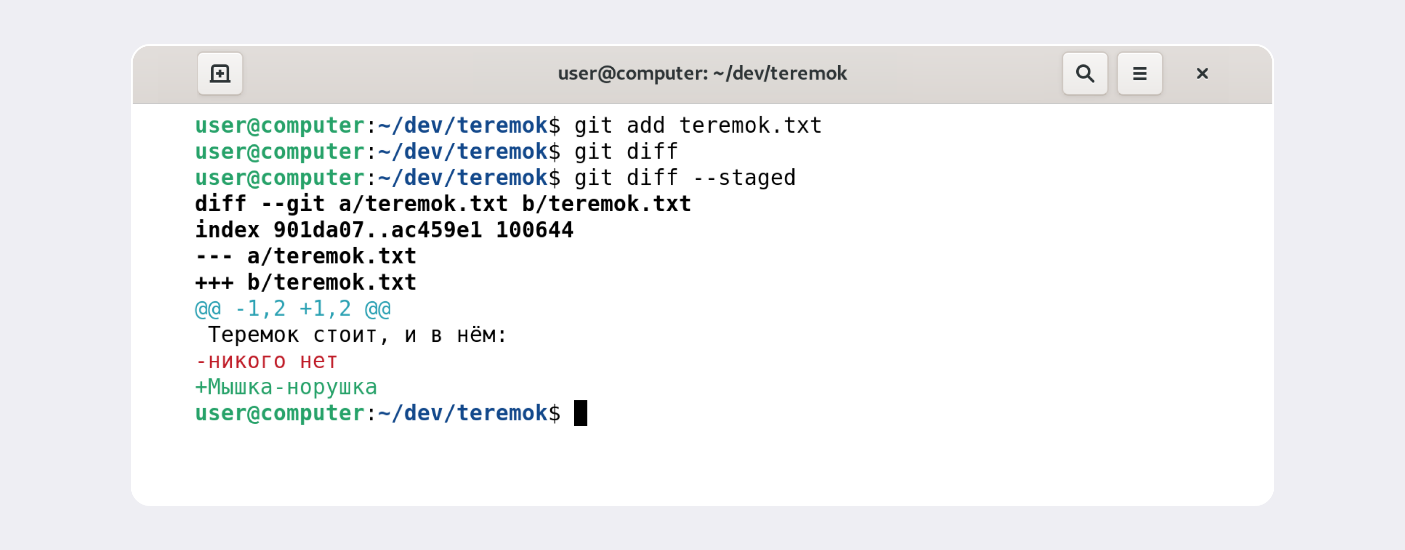
BASH

$ git diff

# команда не выведет ничего!

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.

Скопировать код

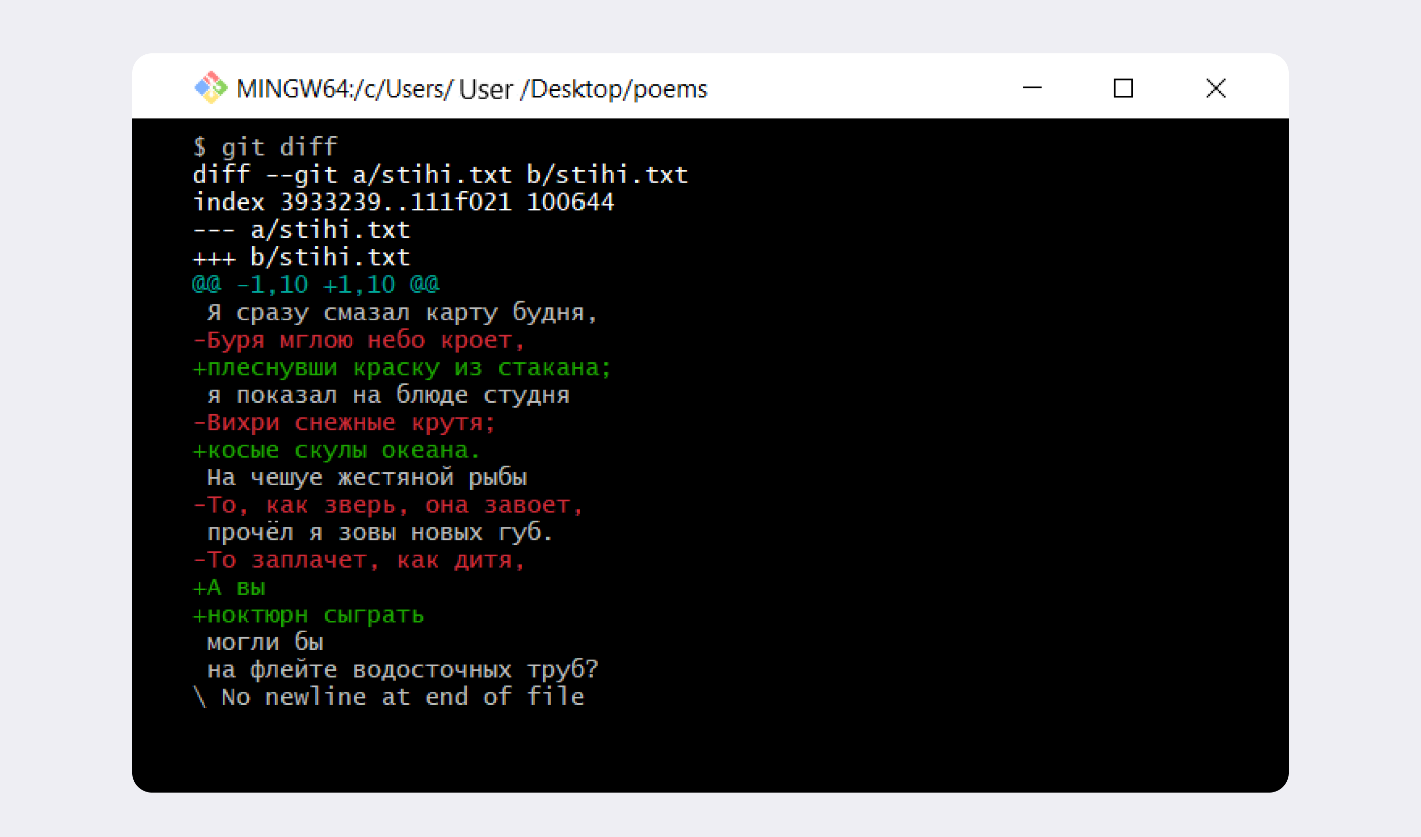
BASH

$ git commit -m "Поселить мышку-норушку"

Готово! В следующем уроке вы допишете сказку и научитесь сопоставлять коммиты.

Начало формы

Изучите вывод команды git diff.



Выберите верные утверждения.



stihi.txt/a — это исходная версия файла.

Посмотрите внимательнее на название файла: мы хотели вас запутать.



В файл добавили шесть новых строк.

Шесть строк остались неизменными.

Правильный ответ

В файл добавили четыре строки.

В нашем примере они выделены зелёным. Также не забывайте обращать внимание на символы + и -.

Правильный ответ

a/stihi.txt — это исходная версия файла.

Исходная версия файла сравнивается с текущей.



Для сравнения использована одна строка, начиная с 1010-й. Об этом говорит выражение @@ -1,10 +1,10 @@.

Наоборот: использовано 1010 строк, начиная с первой.

Правильный ответ

Для сравнения использовано десять строк, начиная с первой. Об этом говорит выражение @@ -1,10 +1,10 @@.

Это те строки, которые попали в сравнение.



b/stihi.txt — это исходная версия файла.

b/stihi.txt — текущая версия файла.

Конец формы

Тадам! Теперь вы можете просматривать изменения в файлах. Если коротко:

* Команда git diff сравнит последнюю закоммиченную версию файла с той, что находится в состоянии modified.
* Команда git diff --staged покажет изменения в staged-файлах относительно последних закоммиченных версий.

**Сопоставляем коммиты**

В этом уроке вы допишете сказку «Теремок», а также научитесь сопоставлять коммиты. Если вдруг на каком-то этапе работы возникнет проблема, это поможет разобраться, что конкретно изменилось в одном коммите по сравнению с другим.

**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.

Скопировать код

BASH

$ echo "Привет!"

Привет!

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 код

BASH

$ cat file.txt

Первая строка файла

$ echo "Вторая строка файла" >> file.txt

$ cat file.txt

Первая строка файла

Вторая строка файла

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.

Скопировать код

BASH

$ cat file.txt

Первая строка файла

$ echo "Новая строка" > file.txt

$ cat file.txt

Новая строка

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.

Скопировать код

BASH

$ echo "Лягушка-квакушка" >> teremok.txt

$ git add teremok.txt

$ git commit -m "Поселить лягушку-квакушку"

# пропускаем вывод команды commit

$ echo "Зайчик-побегайчик" >> teremok.txt

$ git add teremok.txt

$ git commit -m "Поселить зайчика-побегайчика"

$ echo "Лисичка-сестричка" >> teremok.txt

$ git add teremok.txt

$ git commit -m "Поселить лисичку-сестричку"

$ echo "Волчок -- серый бочок" >> teremok.txt

$ git add teremok.txt

$ git commit -m "Поселить волчка -- серого бочка"

# Пришла очередь медведя косолапого.

# Как известно, конструкция теремка не была рассчитана на массу медведя

# и теремок развалился.

# Обратите внимание на одинарный символ >

$ echo "Теремок развален" > teremok.txt

$ git add teremok.txt

$ git commit -m "Попытаться поселить медведя косолапого"

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

Скопировать код

BASH

$ git log --oneline

48fe3dc (HEAD -> master) Попытаться поселить медведя косолапого

318ebdb Поселить волчка -- серого бочка

470c115 Поселить лисичку-сестричку

e4ede9c Поселить зайчика-побегайчика

31cf9b3 Поселить лягушку-квакушку

d7d797a Поселить мышку-норушку

1c29af6 Исходное состояние теремка

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.

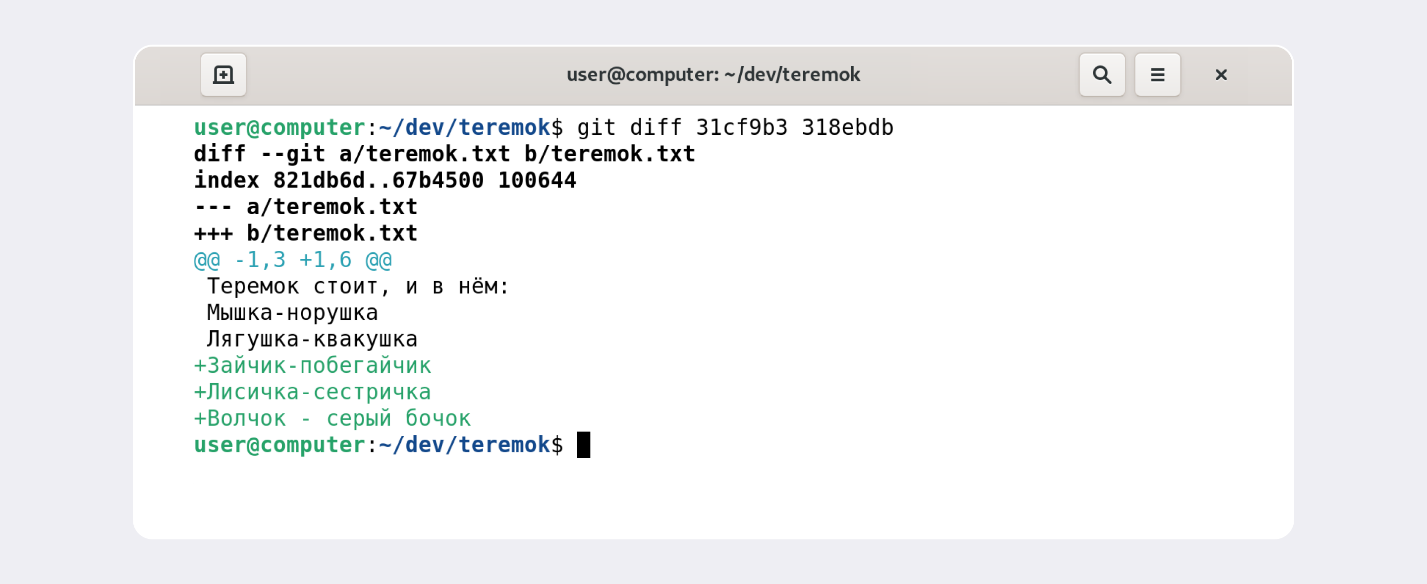
Например, чтобы получить максимально сокращённую версию сказки, можно сравнить состояние файлов на момент первого коммита (у нас это 1c29af6) и последнего (у нас 48fe3dc). Ваши хеши могут отличаться.

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.

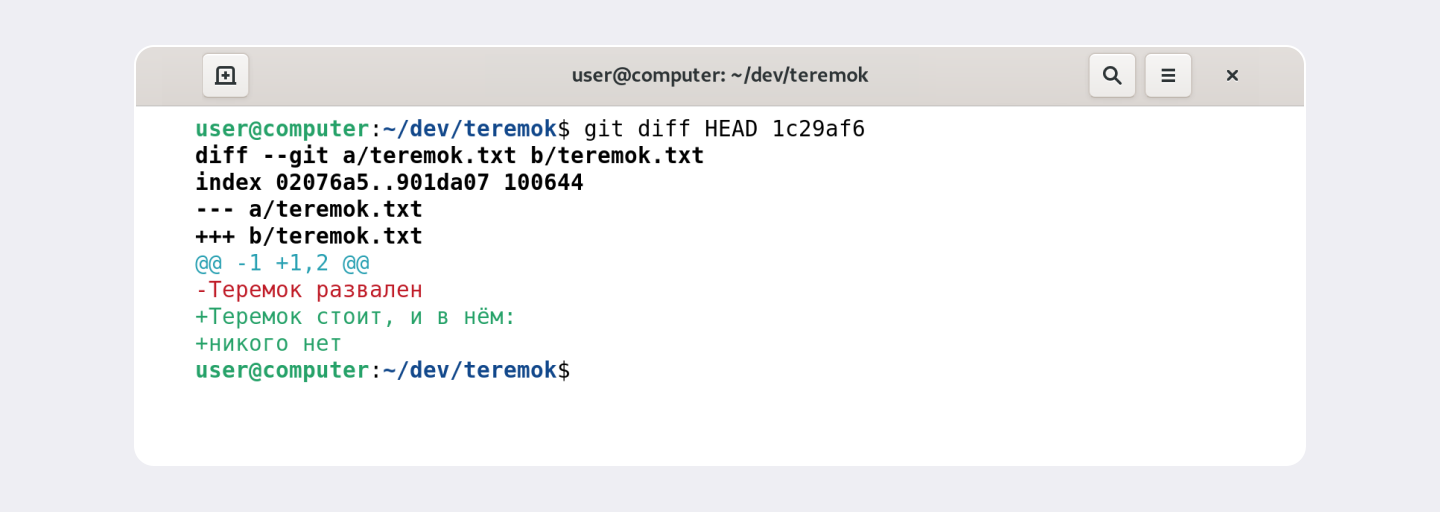


Подселились: зайчик-побегайчик, лисичка-сестричка и волчок — серый бочок. Чёрные строки без знаков + или - — это «контекст», его git diff выводит только для того, чтобы стало понятно, что находится рядом с изменёнными (зелёными и красными) строками.

**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.



Готово! Всё равно, как если бы мы рассказали историю задом наперёд.

Вот и сказке конец, а кто с нами коммитил — молодец!

Начало формы

Выберите верные утверждения о команде git diff.



git diff --log показывает последние изменения в истории коммитов.

Такой команды в Git нет.

Правильный ответ

git diff показывает изменения, которые не были добавлены в staged.

Посмотреть изменения с помощью git diff можно, только если они не были проиндексированы или закоммичены.

Правильный ответ

git log --oneline показывает первые несколько символов хеша и сообщение коммита.

Этой командой можно воспользоваться, когда не нужен полный хеш.



git diff --oneline показывает разницу одной строкой.

В Git нет такой команды.

Неправильный ответ

git log --oneline выводит одну строку: краткое описание последнего коммита.

Эта команда выводит все коммиты, по одному на строку.

Тоже правильный ответ

git diff --staged показывает, что изменилось в проиндексированных файлах.

Ключ --staged говорит системе, что нужно смотреть изменения, которые были добавлены в staged.



git diff --commited показывает, что изменилось в закоммиченных файлах.

Ключа --commited в системе контроля версий нет.

Правильный ответ

git diff a9928ab 11bada1 покажет различия коммитов с хешами a9928ab и 11bada1.

В зависимости от того, в каком порядке переданы коммиты, это будут или внесённые правки, или информация о том, что нужно «откатить».

Конец формы

На этом всё про git diff! Эта команда поможет узнать, какие строки и в каких файлах изменились. Может быть полезно при поиске ошибок или чтобы разобраться, откуда появилась та или иная строка. В этом уроке вы также познакомились с командой git diff <коммит1> <коммит2>. С её помощью удобно сравнивать изменения в двух коммитах.

# Игнорирование файлов в Git

Часто бывает так, что в папке-репозитории есть файлы, для которых не нужно хранить историю изменений. Например:

* macOS иногда создаёт скрытый файл .DS\_Store для хранения настроек папки. К вашему проекту он, скорее всего, никакого отношения не имеет.
* В Git не принято коммитить результаты компиляции исходного кода, то есть получившиеся исполняемые файлы.
* Среды разработки (вроде IntelliJ IDEA) могут создавать папку с вашими личными настройками проекта. Если добавить её в репозиторий, то среда разработки других участников проекта может загрузить ваши настройки и начать вести себя странно.

Чтобы Git игнорировал такие файлы и не пытался добавить их в репозиторий, нужно создать файл .gitignore (от англ. ignore — «игнорировать») и записать в него названия игнорируемых файлов. В этом уроке разберём, как это сделать.

## Как заполнить .gitignore

С точки зрения Git .gitignore — это обычный текстовый файл. Его добавляют в корень репозитория и тоже коммитят.

В простейшем случае в .gitignore указывают все файлы, которые нужно игнорировать (по одному имени на строку). Но часто удобнее использовать **шаблоны**. Шаблон, или правило, — это способ указать сразу на несколько файлов с однотипными названиями.

💡 Правила из .gitignore применяются только к новым (untracked) файлам. Если файл уже попал в staging area или в коммит, то правила на него не распространяются.

Разберём подробнее формат файла .gitignore, какие в нём могут встречаться строки и как выглядят шаблоны.

### Комментарий

Если строка начинается с #, то это комментарий, и .gitignore не будет его учитывать.

Скопировать код

BASH

# вот так можно писать комментарии;

# они ничего не значат для .gitignore,

# но они могут быть полезны, чтобы понять, зачем было добавлено то или иное правило

### Просто название файла

Допустим, нужно, чтобы Git игнорировал все файлы .DS\_Store. Для этого достаточно добавить в .gitignore строку с названием файла.

Скопировать код

BASH

# для macOS

.DS\_Store

В таком случае Git будет игнорировать файлы с именем .DS\_Store, причём не только в корне репозитория, но и во всех вложенных папках.

### ****Звёздочка (****\*****)****

Символ звёздочки (\*) соответствует любой строке, включая пустую. Если такой символ используется в шаблоне в .gitignore, значит, файл будет проигнорирован вне зависимости от того, что будет на месте звёздочки.

Скопировать код

BASH

# игнорировать все файлы, которые заканчиваются на .jpeg

\*.jpeg

# игнорировать все файлы "tmp" во всех подпапках папки docs

docs/\*/tmp

Теперь Git будет игнорировать все файлы, которые заканчиваются на .jpeg — пригодится тем, кто не любит картинки. А также все временные файлы tmp (от англ. ***t***e***mp***orary — «временный») в подпапках папки docs. Например, Git проигнорирует файл docs/current/tmp.

💡 Если задать правило, которое состоит только из звёздочки, Git будет игнорировать все файлы. Это происходит потому, что под звёздочку подходит любое имя файла.

Скопировать код

BASH

# странное, но возможное правило

# "игнорировать все файлы"

\*

### ****Вопросительный знак (****?****)****

Вопросительный знак ? соответствует одному любому символу.

Скопировать код

BASH

file?.txt

Если сохранить такую запись в .gitignore, то будут проигнорированы, например, файлы fileA.txt и file1.txt. А вот файл file12.txt не будет проигнорирован, потому что в его названии два символа после file, а не один.

### Квадратные скобки ([…])

Квадратные скобки, как и вопросительный знак, соответствуют одному символу. При этом символ не любой, а только из списка, который указан в скобках.

Скопировать код

BASH

# игнорировать файлы file0.txt, file1.txt и file2.txt

# при этом не игнорировать file3.txt, file4.txt, ...

file[0-2].txt

В скобках можно либо перечислить символы ([abc]), либо задать диапазон ([a-z]).

### Слеш (/)

Косая черта, или слеш (/), указывает на каталоги. Если шаблон в .gitignore начинается со слеша, то Git проигнорирует файлы или каталоги только в корневой директории.

Скопировать код

BASH

# игнорировать todo.txt в корне репозитория

/todo.txt

# для сравнения: spam.txt будет игнорироваться во всех папках

spam.txt

Теперь файл todo.txt в корневом каталоге будет проигнорирован. При этом, например, файл subdir/todo.txt по-прежнему отслеживается.

Если шаблон заканчивается слешем, то правило применится только к папке.

Скопировать код

BASH

# игнорировать папку build

build/

Обратите внимание: если build — это папка, то она будет проигнорирована. Если build — обычный файл, то он не подпадёт под правило и не будет игнорироваться.

### Парные звёздочки (\*\*)

Функция парных звёздочек (\*\*) похожа на функцию одинарной (\*). Отличие в том, как они работают с вложенными папками. Двойная звёздочка может соответствовать любому количеству таких папок (в том числе нулю). Одинарная может соответствовать только одной.

Скопировать код

BASH

# игнорировать файлы "docs/current/tmp", "docs/old/tmp",

# а также "docs/old/saved/a/b/c/d/tmp"

# и даже "docs/tmp", потому что ноль вложенных папок тоже подходит

docs/\*\*/tmp

# игнорировать только "docs/current/tmp" и "docs/old/tmp"

# файл "docs/old/saved/a/b/c/d/tmp" не попадает в правило

docs/\*/tmp

💡 Для двойной звёздочки верно то же самое, что и для одной: если задать правило \*\*, то будут проигнорированы все файлы.

### Восклицательный знак (!)

Любое правило в файле .gitignore можно инвертировать с помощью восклицательного знака (!).

Скопировать код

BASH

# игнорировать все JPEG-файлы

\*.jpeg

# но только не мем с Doge

!doge.jpeg

Теперь файл doge.jpeg будет отслеживаться, хотя остальные jpeg-файлы будут проигнорированы. Такие правила удобны для добавления исключений из других правил .gitignore.

### Пример файла .gitignore

Содержание .gitignore может быть таким.

Скопировать код

BASH

# игнорировать все файлы в каталоге build

build/

# игнорировать все .log файлы

\*.log

# не игнорировать \*.log файлы в examples

# потому что это пример для документации

!examples/\*\*/\*.log

## .gitignore и git status

Игнорируемые файлы не отображаются в выводе команды git status, иначе они бы засоряли вывод.

Если всё же нужно отобразить все игнорируемые файлы, то это можно сделать с помощью ключа --ignored: git status --ignored. В таком случае в выводе git status появится раздел Ignored files.

## Задание для самостоятельной работы

Многие научные статьи оформляют с помощью **TeX**. Этот инструмент принимает на вход файлы с расширением .tex и выдаёт в качестве результата красиво оформленный .pdf. Но есть подвох: в процессе работы TeX также создаёт много временных файлов с расширениями вроде .aux, .lof, .lot и так далее.

Составьте файл .gitignore так, чтобы Git игнорировал все файлы, кроме .tex и .pdf. Вы можете свериться с авторским решением, но сначала рекомендуем попробовать выполнить задание самостоятельно.

Показать авторское решение

Замечательно! Игнорирование файлов — механизм, который нельзя игнорировать. Вот что важно помнить:

* Если нужно, чтобы Git игнорировал какие-то файлы, стоит составить файл .gitignore.
* Посмотреть, что игнорируется, можно с помощью команды git status --ignored.
* Сам файл .gitignore — это обычный файл в репозитории. Его тоже стоит закоммитить.
* Шаблонов много, но их легко найти в интернете вместе с примерами использования.

**Практическая работа №4. Исследуем историю коммитов**

Пришло время попрактиковаться! А финальный босс этого модуля — репозиторий movies, который создал кинолюб Василий. Ваша задача — исследовать историю проекта и выполнить задания.

В процессе вам понадобятся команды git log и git diff. Эти команды не раз пригодятся в реальной работе — для анализа текущего состояния репозитория и поиска ошибок или чтобы просто разобраться, откуда в проекте появилось изменение. Вперёд!

**Как всё устроено в репозитории**

Разработчик Василий любит две вещи: Git и фильмы. Хороших фильмов так много, что наш герой решил завести список кинокартин, которые нужно обязательно посмотреть. Для этого Василий совместил приятное с полезным: создал Git-репозиторий с одним файлом-списком movies.txt.

Василий много работал, и времени на плохие фильмы у него не было. Поэтому он выбирал лучшее: пару кинолент из номинированных на «Оскар» и пару претендентов на главный приз Каннского фестиваля.

Когда Василий узнавал о новом достойном фильме, он добавлял его в список в movies.txt. После того как фильм был отсмотрен, Василий убирал его из списка. После каждого действия он делал коммит.

Герой начал список в 20112011 году, а к 20122012 году прекратил вести файл — решил просто добавлять нужные фильмы в закладки на киносервисе. Теперь репозиторий попал к вам.

**Что нужно сделать**

Скачайте репозиторий Василия на свой локальный компьютер.

[movies\_repo.zip](https://code.s3.yandex.net/BasicsOfGit/movies-repo.zip?etag=8fcb3ddde11fcef6d86c27991be06e2e)

Разархивируйте репозиторий и перейдите в папку с проектом через консоль. Изучите историю и содержимое коммитов и выполните задания ниже.

Начало формы

Сколько всего фильмов посмотрел Василий за 20112011 и 20122012 годы?



88

У нашего героя хватило времени на восемь фильмов и ещё один.



Тоже правильный ответ

99

Именно столько фильмов удалось посмотреть Василию!



1010

Василий очень хотел посмотреть десять фильмов, но не успел.



1111

К сожалению, у нашего героя не было времени на одиннадцать фильмов.

Конец формы

💡 **Как можно действовать?**

По названиям и содержимому коммитов можно увидеть, что коммиты с сообщениями Oscar #… и Cannes #… — это те, в которых фильмы добавляются в список. А те коммиты, в сообщении которых стоит дата, говорят, когда Василий посмотрел фильм.

Можно либо посчитать суммарное количество добавленных строк в коммитах 6c537a6, c985f54, 902c538 и f03339d, либо посчитать количество коммитов, в сообщении которых указана только дата.

Содержимое коммита можно проверить командой git diff <предыдущий коммит> <текущий коммит>.

Начало формы

Когда Василий посмотрел фильм «Король говорит»?



2929 июня 20122012 года (2012-06-29)

В этот день Василий посмотрел «Прислугу».



Неправильный ответ

2727 мая 20112011 года (2011-05-27)

2727 мая герой посмотрел «Меланхолию».



Тоже правильный ответ

2626 августа 20112011 года (2011-08-26)

За окном было мокро и холодно, и Василий решил посмотреть фильм о превозмогании.



3131 декабря 20112011 года (2011-12-31)

В этот день Василий посмотрел «Древо жизни».

Конец формы

💡 **Как можно действовать?**

Фильмы добавляются в список в коммитах Oscar #… и Cannes #…. Таких всего четыре, поэтому можно проверить содержимое их всех.

Фильм был добавлен в коммите 6c537a6 с сообщением Oscar #83. Можно сначала подбором найти любой коммит, в котором уже нет строки The King’s Speech / Король говорит. А затем двигаться назад во времени — пока не найдётся коммит, в котором Василий вычеркнул фильм из списка.

Содержимое коммита можно проверить командой git diff <предыдущий коммит> <текущий коммит>.

Начало формы

Какой фильм Василий посмотрел последним?



Правильный ответ

*The Artist* («Артист»)

Верно: это было 2828 декабря 20122012 года.



*Inception* («Начало»)

Этот фильм Василий посмотрел 44 марта 20112011 года.



*The Hateful Eight* («Омерзительная восьмёрка»)

Этого фильма нет в списке Василия.



*The Help* («Прислуга»)

Это было 2929 июня 20122012 года.

Конец формы

💡 **Как можно действовать?**

По умолчанию коммиты в истории упорядочены так, что наверху списка — более поздние по дате. Нужно найти последний коммит, в котором вычеркнуто название фильма. Это коммит 2dd1ad5; в нём Василий убрал строчку The Artist / Артист.

Это был крепкий орешек! По этому упражнению можно заметить, насколько проще было бы отвечать на вопросы, если бы сообщения коммитов были более информативными. Например: 2012-06-29: Вычеркнуть "Прислугу" вместо 2012-06-29. Или Oscar #83: Добавить 3 фильма вместо Oscar #83.

Не расстраивайтесь, если что-то не получилось: попробуйте исследовать репозиторий ещё раз, опираясь на наши пояснения.

git diff <начало модуля> <конец модуля>

-страх +знания

**Шпаргалка. Начало работы с Git**

Чтобы вам было проще запомнить все команды, о которых шла речь в этом модуле, мы собрали их в одном месте.

**Инициализация репозитория**

git init (от англ. ***init****ialize*, «инициализировать») — инициализируй репозиторий.

**Синхронизация локального и удалённого репозиториев**

git remote add origin https://github.com/YandexPracticum/first-project.git (от англ. *remote*, «удалённый» + *add,* «добавить») — привяжи локальный репозиторий к удалённому с URL https://github.com/YandexPracticum/first-project.git;

git remote -v (от англ. ***v****erbose*, «подробный») — проверь, что репозитории действительно связались;

git push -u origin main (от англ. *push*, «толкать») — в первый раз загрузи все коммиты из локального репозитория в удалённый с названием origin.

💡 Ваша ветка может называться master, а не main. Подправьте команду, если это необходимо.

git push (от англ. *push*, «толкать») — загрузи коммиты в удалённый репозиторий после того, как он был привязан с помощью флага -u.

**Подготовка файла к коммиту**

git add todo.txt (от англ. *add*, «добавить») — подготовь файл todo.txt к коммиту;

git add --all (от англ. *add*, «добавить» + *all*, «всё») — подготовь к коммиту сразу все файлы, в которых были изменения, и все новые файлы;

git add . — подготовь к коммиту текущую папку и все файлы в ней.

**Создание и публикация коммита**

git commit -m "Комментарий к коммиту." (от англ. *commit,* «совершать», фиксировать» + ***m****essage,* «сообщение») — сделай коммит и оставь комментарий, чтобы было проще понять, какие изменения сделаны;

git push (от англ. *push*, «толкать») — добавь изменения в удалённый репозиторий.

**Просмотр информации о коммитах**

git log (от англ. *log*, «журнал [записей]») — выведи подробную историю коммитов;

git log --oneline (от англ. *log*, «журнал [записей]» + *oneline*, «одной строкой») — покажи краткую информацию о коммитах: сокращённый хеш и сообщение.

**Просмотр состояния файлов**

git status (от англ. *status*, «статус», «состояние») — покажи текущее состояние репозитория.

**Добавление изменений в последний коммит**

git commit --amend --no-edit (от англ. *amend*, «исправить») — добавь изменения к последнему коммиту и оставь сообщение прежним;

git commit --amend -m "Новое сообщение" — измени сообщение к последнему коммиту на Новое сообщение.

💡 Выйти из редактора Vim: нажать Esc, ввести :qa!, нажать Enter.

**«Откат» файлов и коммитов**

git restore --staged hello.txt (от англ. *restore*, «восстановить») — переведи файл hello.txt из состояния staged обратно в untracked или modified;

git restore hello.txt — верни файл hello.txt к последней версии, которая была сохранена через git commit или git add;

git reset --hard b576d89 (от англ. *reset*, «сброс», «обнуление» + *hard*, «суровый») — удали все незакоммиченные изменения из staging и «рабочей зоны» вплоть до указанного коммита.

**Просмотр изменений**

git diff (от англ. ***diff****erence*, «отличие», «разница») — покажи изменения в «рабочей зоне», то есть в modified-файлах;

git diff a9928ab 11bada1 — выведи разницу между двумя коммитами;

git diff --staged — покажи изменения, которые добавлены в staged-файлах.

Вы с нами уже два модуля и наверняка успели составить некоторое впечатление о курсе. Надеемся, что вам с нами интересно!