

**МИНОБРНАУКИ РОССИИ**

**федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технологический университет «СТАНКИН»**

**(ФГБОУ ВО «МГТУ «СТАНКИН»)**

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | Институт цифровых интеллектуальных систем | Кафедра  компьютерных систем управления | |

Дисциплина «Основы системного программного обеспечения»

**Отчет по лабораторной работе №\_1\_**

|  |  |  |
| --- | --- | --- |
| **Выполнил**  **студент гр. АДБ-20-08:** | \_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_  *(дата) (подпись)* | **Курнин Ю.А.** |
| **Проверил**  **к.т.н., доцент** | \_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_  *(дата) (подпись)* | **Ковалев И.А.** |

**Москва 2023 г.**

**Содержание**

1. **Цель………………………………………………………………………………………2**
2. **Задачи работы…………………………………………………………………………..2**
3. **Описание и выполнение заданий ……………………………………………………3**

**Цель:** научиться работать с Git.

**Задачи:** создать учетную запись и репозиторий в нём, далее, создавая и изменяя локальные файлы, использовать Git для их синхронизации с репозиторием.

**Индивидуальные задания:**

Перейдем в созданную нами папку в командной строке используя команду cd.

Теперь необходимо проинициализировать эту папку как git репозиторий

Введем в командной строке: git init

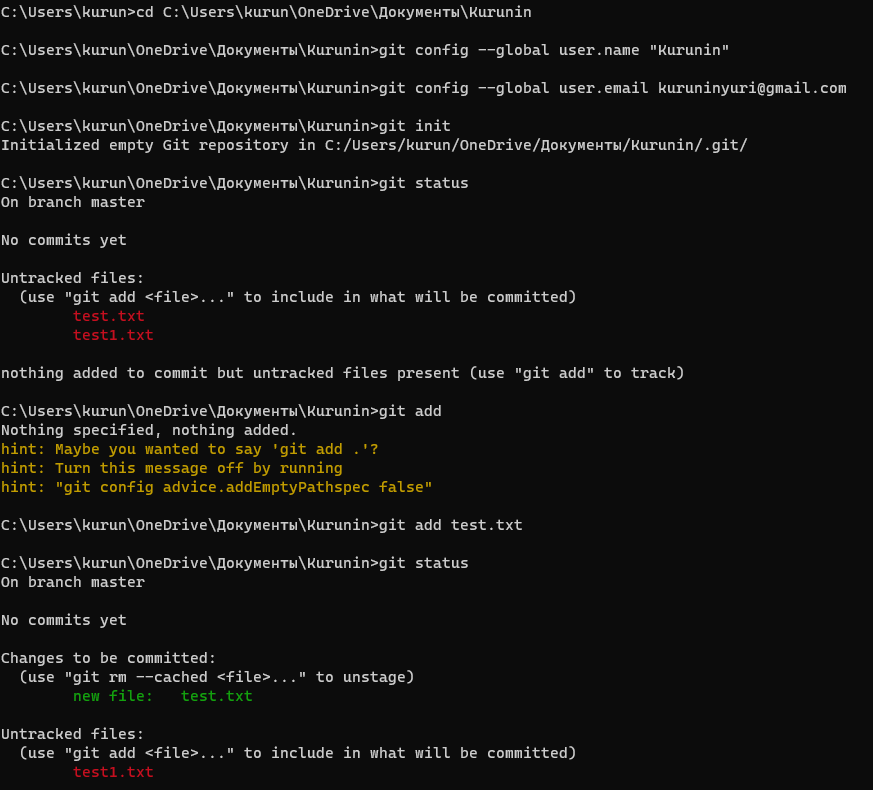
Командная строка вернет нам сообщение, что проинициализирована пустая git директория

Теперь используя проводник windows создадим в своей папке любой текстовый файл и пропишим в него наше имя.

Введем команду: git stаtus и нам отобразиться сообщение, что есть новый файл, но он не отслеживается. Это значит, что файл новый и система еще не знает, нужно ли следить за изменениями в файле или его можно просто игнорировать.

Для того, чтобы начать отслеживать новый файл, нужно его специальным образом объявить. В git есть концепция области подготовленных файлов. Можно представить ее как холст, на который наносят изменения, которые нужны в коммите. Сперва он пустой, но затем мы добавляем на него файлы (или части файлов, или даже одиночные строчки) командой add и, наконец, коммитим все нужное в репозиторий (создаем слепок нужного нам состояния) командой commit.

Фиксация изменений в области заготовленных файлов. Если хотим добавить только один файл, можно написать: git add test.txt Если измененных файлов много, то можно сделать так: git add .

Выполнив нужную команду снова посмотрим статус репозитория

Файл готов к коммиту. Сообщение о состоянии также говорит нам о том, какие изменения относительно файла были проведены в области подготовки — в данном случае это новый файл, но файлы могут быть модифицированы или удалены.

Теперь закомитим наши файлы. Коммит представляет собой состояние репозитория в определенный момент времени как некий опечаток во времени. Коммит отмечается hash суммой и к которому мы можем в любой момент времени вернуться.

Чтобы зафиксировать изменения, нам нужно хотя бы одно изменение в области подготовки (мы как раз создали новый файл): git commit -m "first commit"

Команда создаст новый коммит со всеми изменениями из области подготовки. Считается хорошей практикой делать коммиты часто и всегда писать содержательные комментарии. Вызовем git status и посмотрим, что поменялось.

**Отправка коммита на сервер**

Перед тем как перейти к более продвинутым вещам, рассмотрим, как связать наш локальный репозиторий с облачным репозиторием, который мы создали на github.com Чтобы связать наш локальный репозиторий с репозиторием на GitHub, выполним следующую команду в терминале. git remote add origin \*URL репозитория\*

Проект может иметь несколько удаленных репозиториев одновременно. Чтобы их различать, мы дадим им разные имена. Обычно главный репозиторий называется origin. Можно вызвать команду, для просмотра, к какому проекту мы подключены git remote -v

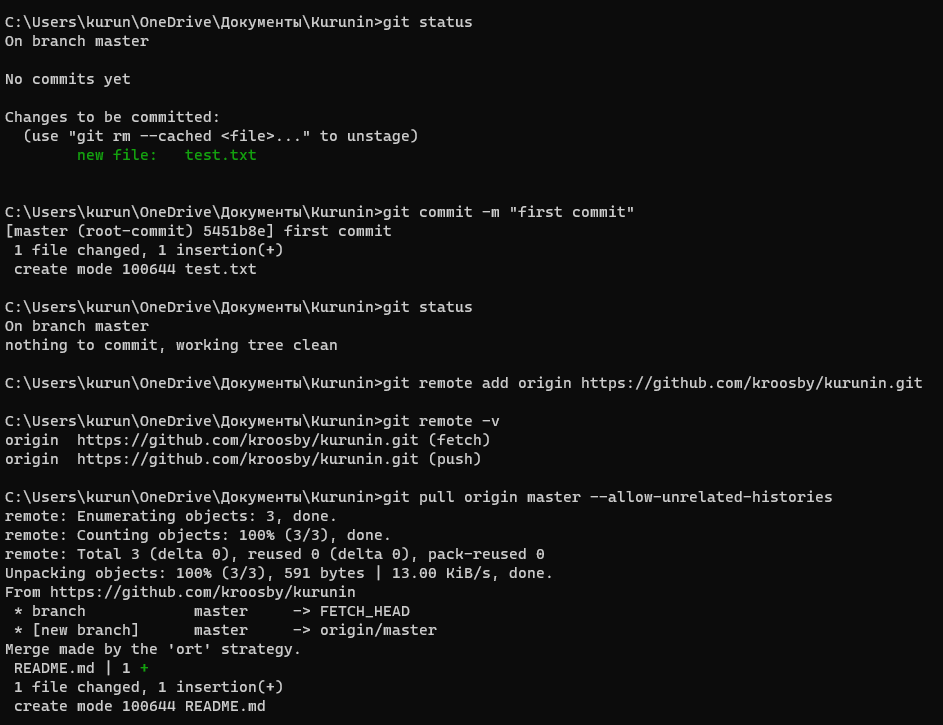
Если вдруг у вас на этом месте показывается несколько репозиториев, то скорее всего верхние команды привели к запутыванию веток, это можно разрешить следующей командой. Также это команда может пригодиться, если у вас возникли ошибки, связанные с unrelated-histories. git pull origin master --allow-unrelated-histories

Запустим git remote -v снова

Если репозиторий указан неверно, его можно удалить командой (но только если он указан неверно) git remote rm origin

После необходимо снова подключить верный репозиторий

Запросим изменения с сервера В нашем случае у нас есть файл и на сервере и локальном репозитории, поэтому нам вначале надо получить все данные с нашего репозитория, а потом уже их заливать на сервер нашими локальными данными.



**Теперь перешлем локальный коммит на сервер**

Теперь отравим коммит на сервер, команда, предназначенная для этого — push.

Она принимает два параметра: имя удаленного репозитория (мы назвали наш origin) и ветку, в которую необходимо внести изменения (master — это ветка по умолчанию для всех репозиториев)

. git push origin master

У нас должно появиться окно, где нужно ввести авторизоваться на github.com

Выбираем вариант использования браузера – надо нажать Sign in with your browser Откроется страница в браузере по умолчанию (если вы использовали для регистрации этот же браузер и не выходили из github.com, то все ок. Если это другой браузер, то залогинемся на github под своим аккантом

Посмотрим наши изменения с использованием команды git log

Видны коммиты по изменениям в файле Readme.md и в файле с именем (first commit), чтобы выйти из просмотра лога надо нажать q

Посмотрим изменения на github.com

Если нажать на «commint», то увидим все наши изенения

**Создание новой ветки**

Во время разработки новой функциональности считается хорошей практикой работать с копией оригинального проекта, которую называют веткой. Ветви имеют свою собственную историю и изолированные друг от друга изменения до тех пор, пока вы не решаете слить изменения вместе. Это происходит по набору причин:

• Уже рабочая, стабильная версия кода сохраняется.  
 • Различные новые функции могут разрабатываться параллельно разными программистами.  
 • Разработчики могут работать с собственными ветками без риска, что кодовая база поменяется из-за чужих изменений.  
 • В случае сомнений, различные реализации одной и той же идеи могут быть разработаны в разных ветках и затем сравниваться.

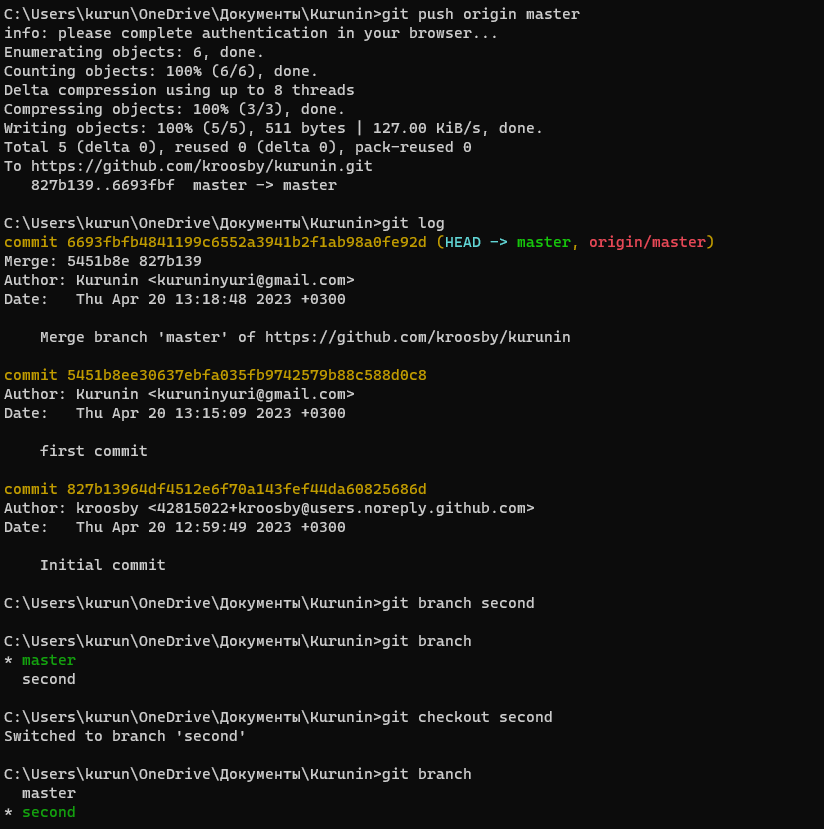
Обычно в проекте создается минимум 2 ветки (branch) – master и ветка для разработки. Ветка master создается сам, а дополнительные ветки создаются по желанию пользователя (локально), заливаются они в удаленный репозиторий уже с учетом доступных прав.

Создадим новую ветку second: git branch second

Можно посмотреть в какой ветки находимся, используя команду git branch

Переключимся на другую ветку, используя команду git checkout second

Создадим новый файл в нашем локальном репозитории и напишетм в нем свою фамилию, добавьте в область подготовленных файлов, закомитем и отправим на сервер. Зайдем на github и посмотрим, что в ветке master нет нашего файла, а в ветке second он присутсует



**Слияние веток**

Переключимся снова на ветку master

Если через проводник windows в папку с проектом, то файла test\_new.txt не будет виден, потому что мы переключились обратно на ветку master, в которой такого файла не существует. Чтобы он появился, нужно воспользоваться merge для объединения веток (применения изменений из ветки second к основной версии проекта)

. git merge second

Если все прошло без ошибок, можно удалить ветку git branch –d second

Давайте немного усложним это задание, предположив, что в двух ветках могут быть одинаковые файлы и над ними работают разные разработчики

Создадим ветку с названием newdev

Переключимся на нее. Добавим в файл с именем отчество. Зафиксируем изменения. Закомитим изменения.

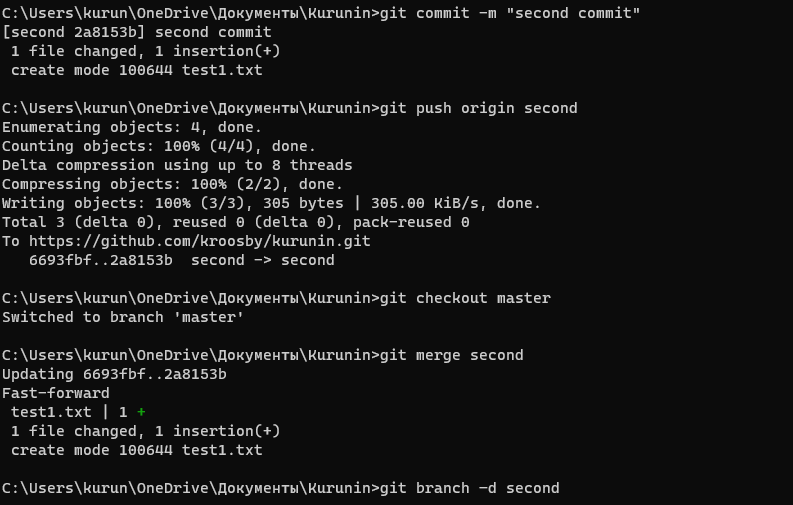
Попробуем переключиться обратно на ветку master и посмотреть файл: в нем только имя, отчество в другой ветке.

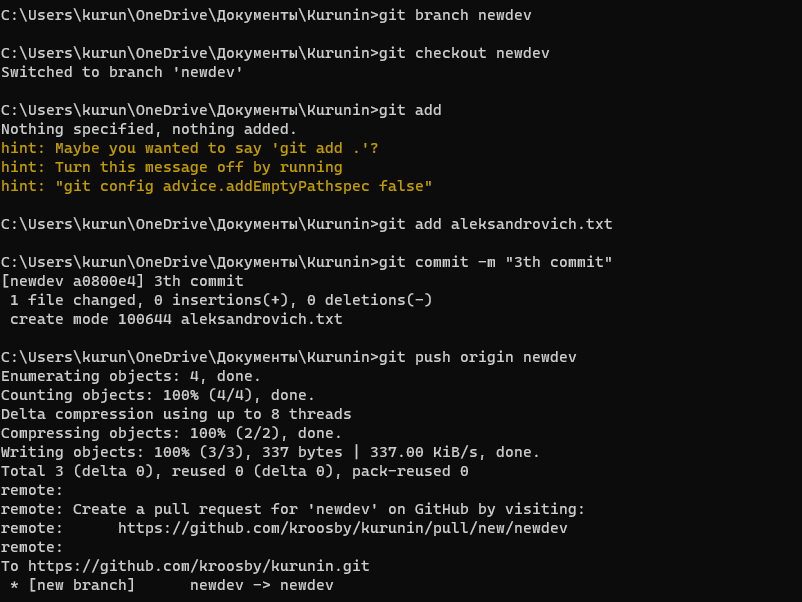
Объединим ветки git merge newdev

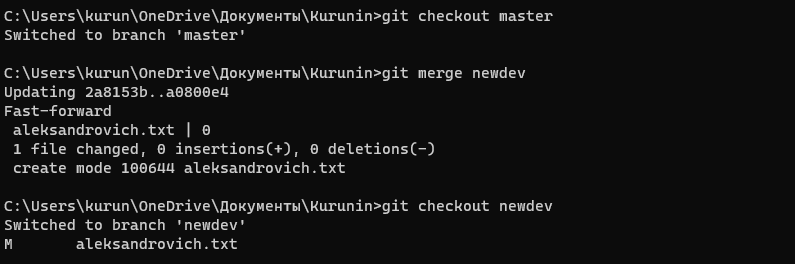
Откроем файл снова, там есть и имя, и отчество

Переключимся на ветку newdev и удалим в отчестве несколько букв. Зафиксируем, закоммитим Переключимся на ветку master, добавим к отчеству несколько букв. Зафиксируем, закоммитим Объединим ветки git merge newdev

Теперь ничего не получится, т.к. есть изменения в обеих ветках







**Просмотр изменений и разрешение конфликтов**

Нужно разрешить конфликты. Наберем команду, для просмотра изменений git diff

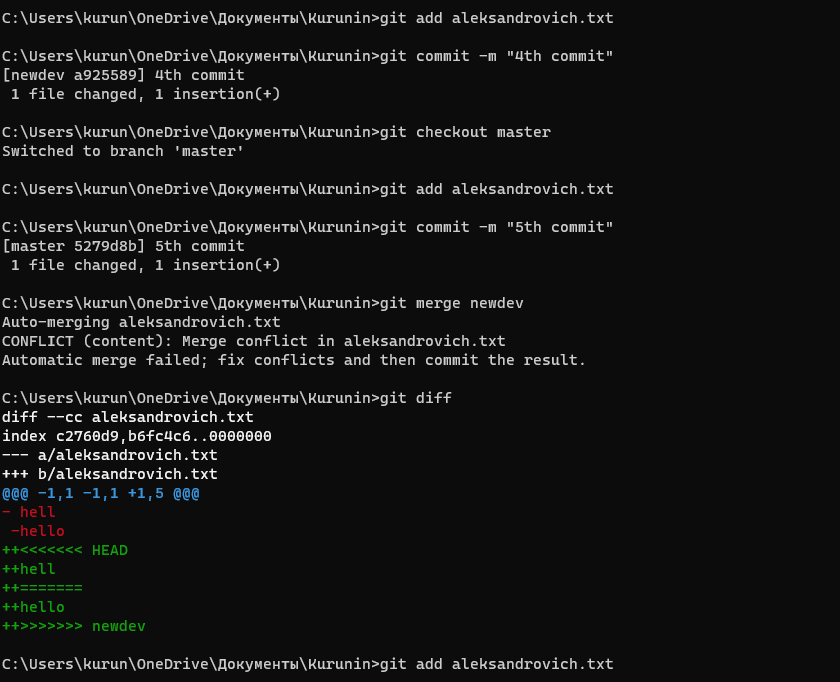
Иногда конфликты исправляются автоматически, но обычно с этим приходится разбираться вручную — решать, какой код остается, а какой нужно удалить. Система не смогла разрешить конфликт автоматически, значит, это придется сделать разработчикам. Приложение отметило строки, содержащие конфликт:

<<<<<<< HEAD \* \* =======

\* \* >>>>>>> newdev

Над разделителем ======= мы видим последний (HEAD) коммит, а под ним — конфликтующий. Таким образом, мы можем увидеть, чем они отличаются и решать, какая версия лучше. Или вовсе написать новую. В этой ситуации мы так и поступим, перепишем все, удалив разделители (HEAD, ===, <<>>), и дадим git понять, что закончили.

Процесс может быть довольно утомительным и может быть очень сложным в больших проектах. Многие разработчики предпочитают использовать для разрешения конфликтов клиенты с графическим интерфейсом. Выбираем нужную строку (какую строку оставить, может вообще все хотим оставить), удаляем разделители, сохраняем файл, фиксируем изменения, коммитим, отправляем на сервер Удалиv ветку newdev



**Удаление веток на сервере**

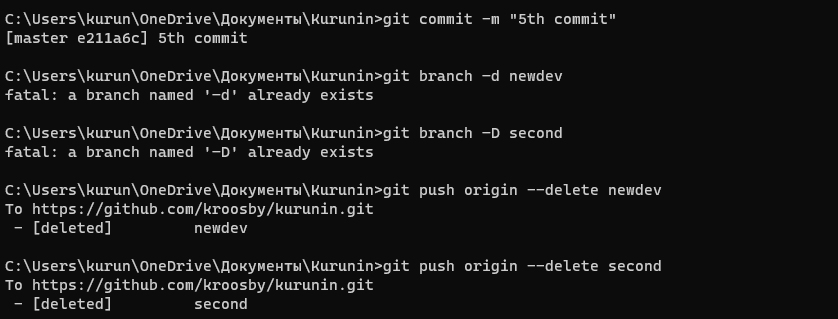
Если у вас есть несмерженная ветка, но вы хотите ее удалить, необходимо использовать команду: git branch –D second

Кстати, если посмотреть на github, то наши ветки там остались:

Давайте удалим их из github:

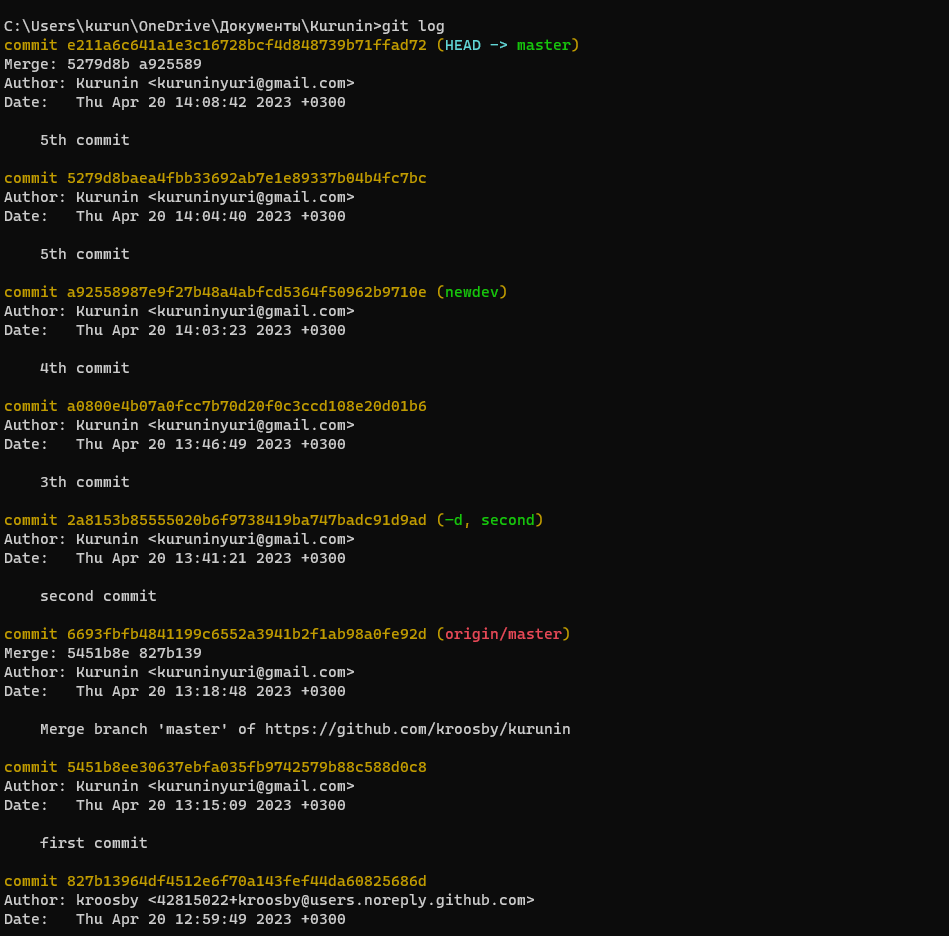
git push origin --delete newdev

git push origin --delete second

Все ветки удалены 

**Возврат к предыдущему состоянию**

Гит позволяет вернуть выбранный файл к состоянию на момент определенного коммита. Это делается уже знакомой нам командой checkout, которую мы ранее использовали для переключения между ветками. Но она также может быть использована для переключения между коммитами.

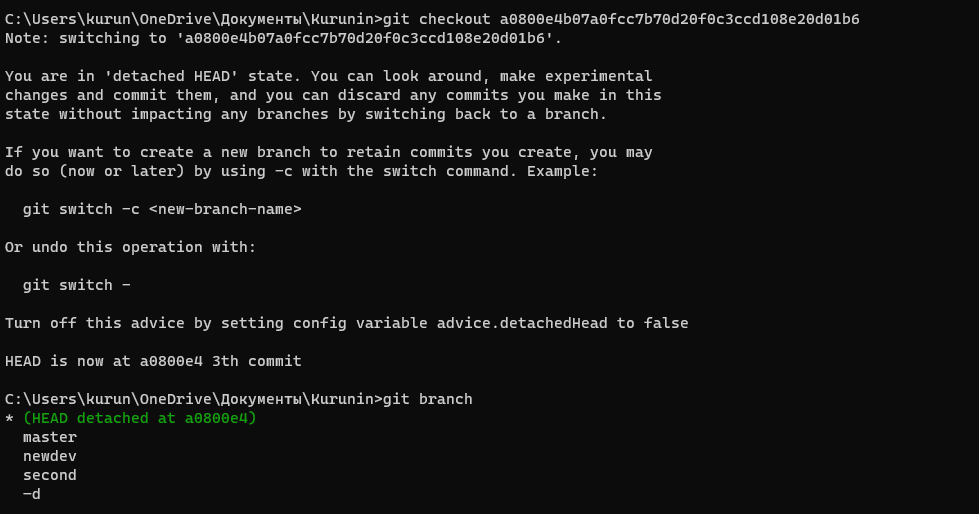
Чтобы посмотреть все комиты, можно использовать команду: git log

Выберем любой коммит, на который хотим откатится, достаточно указать его первые несколько символов (перед выполнением посмотрите свои файлы, чтобы запомнить, что там находится):

git checkout [hash commit]

Если посмотреть файлы теперь, то увидим, что они поменялись – мы вернулись назад

Чтобы это произошло создалась псевдо-ветка начинающаяся на этом коммите, посмотрим ветки и увидим:



**Исправление коммита**

Возможно ситуация, что вы закрепили файлы (поменяйте что-то в файле и закрепите их, вызовите git status), но еще не коммитили, хотите убрать файлы из области закрепления, вызовите команду: git reset HEAD

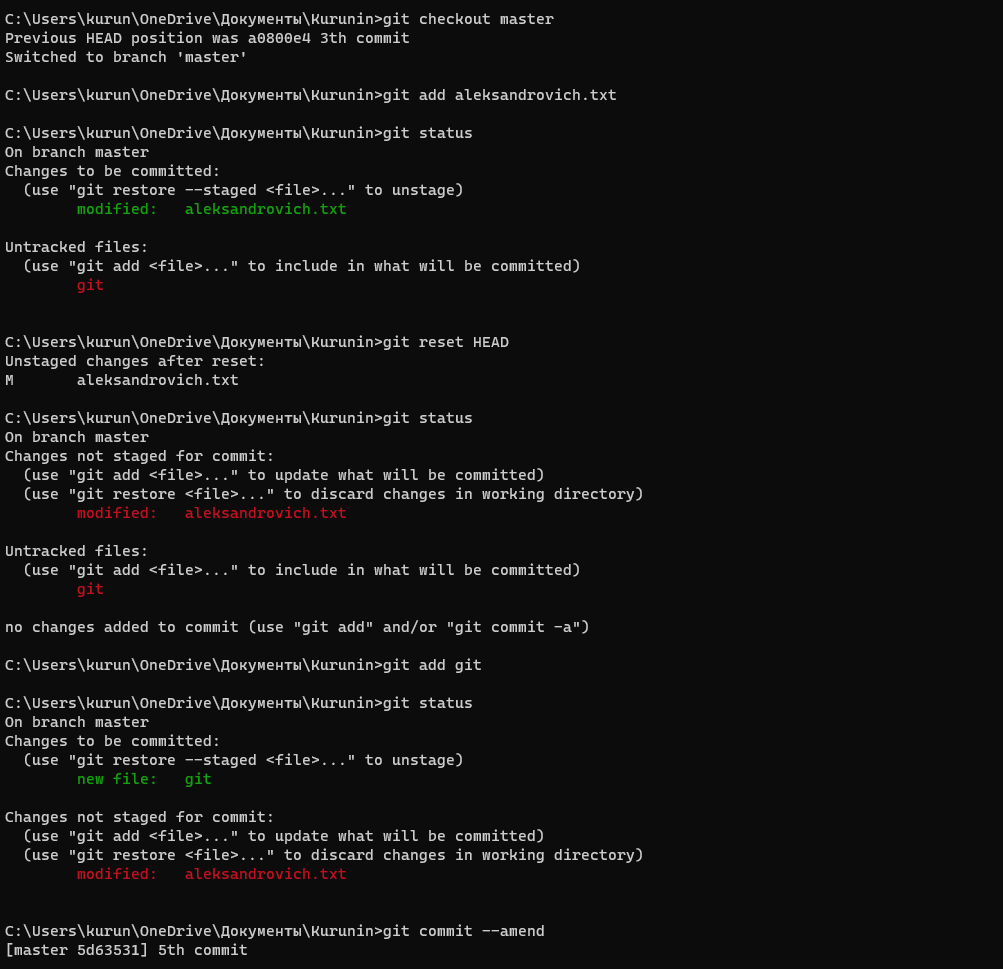
Ваши файлы останутся такими же, но уйдут из области закрепления (вызовите git status) и снова будет показано, что есть измененные файлы.

Теперь снова добавим в область закрепления и закомитим, но еще не отправляем на сервер, можно вызвать ту же команду и тогда все, что осталось незакомиченным будет удалено.

Если хотим изменить комментарий можно вызвать команду:

git commit --amend

Откроется консоль в редакторе Vi, чтобы начать что-то менять, надо нажать клавишу Insert, когда изменения в комментарий внесены, необходимо снова нажать Insert. Чтобы сохранить изменения, нажмите Esc, потом нажмите двоеточие (появится внизу экрана), напишите w! и нажмите Enter. Для выхода снова нажмем двоеточие, введите q! и нажмем Enter.



Cooperation.

