

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Московский государственный технологический университет «СТАНКИН»

(ФГБОУ ВО МГТУ «СТАНКИН»)

Институт цифровых интеллектуальных систем

Кафедра «Компьютерные системы управления»

Образовательная программа 15.03.04

«Автоматизация технологических процессов и производств»

Дисциплина «Основы системного программного обеспечения»

Отчёт по лабораторной работе №1

«Работа с системами контроля версий на примере GitHub»

|  |
| --- |
|  |
| (дата) |

|  |
| --- |
|  |
| (подпись) |

Выполнил: Кузьмич К.С.

студент группы: АДБ 21-06

|  |
| --- |
|  |
| (дата) |

|  |
| --- |
|  |
| (подпись) |

Проверил: Ковалев И.А.

к.т.н., доцент

Москва 2024 г.

Оглавление

[***Цель работы:*** 3](#_Toc168510306)

[***Краткая теория:*** 3](#_Toc168510307)

[Основные задачи работы: 5](#_Toc168510308)

[Ход работы: 5](#_Toc168510309)

[Вывод: 22](#_Toc168510310)

[Список использованных литературы: 23](#_Toc168510311)

***Лабораторная работа №1***

**«Работа с системами контроля версий на примере GitHub»**

***Цель работы:***

Освоение навыков работы с системами контроля версий на примере GitHub.

***Краткая теория:***

Система контроля версий (СКВ) — это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов. Для примеров будем использовать исходные коды программ, но на самом деле под версионный контроль можно поместить файлы практически любого типа.

Главное отличие Git'а от любых других СКВ (например, Subversion и ей подобных) — это то, как Git смотрит на свои данные. В принципе, большинство других систем хранит информацию как список изменений (патчей) для файлов. Эти системы (CVS, Subversion, Perforce, Bazaar и другие) относятся к хранимым данным как к набору файлов и изменений, сделанных для каждого из этих файлов во времени. Git считает хранимые данные набором слепков небольшой файловой системы. Каждый раз, когда вы фиксируете текущую версию проекта, Git, по сути, сохраняет слепок того, как выглядят все файлы проекта на текущий момент. Ради эффективности, если файл не менялся, Git не сохраняет файл снова, а делает ссылку на ранее сохранённый файл.

Это важное отличие Git'а от практически всех других систем контроля версий. Из-за него Git вынужден пересмотреть практически все аспекты контроля версий, которые другие системы переняли от своих предшественниц. Git больше похож на небольшую файловую систему с невероятно мощными инструментами, работающими поверх неё, чем на просто СКВ.

Для совершения большинства операций в Git'е необходимы только локальные файлы и ресурсы, т.е. обычно информация с других компьютеров в сети не нужна. Если вы пользовались централизованными системами, где практически на каждую операцию накладывается сетевая задержка, вы, возможно, подумаете, что боги наделили Git неземной силой. Поскольку вся история проекта хранится локально у вас на диске, большинство операций кажутся практически мгновенными.

К примеру, чтобы показать историю проекта, Git'у не нужно скачивать её с сервера, он просто читает её прямо из вашего локального репозитория. Поэтому историю вы увидите практически мгновенно. Если вам нужно просмотреть изменения между текущей версией файла и версией, сделанной месяц назад, Git может взять файл месячной давности и вычислить разницу на месте, вместо того чтобы запрашивать разницу у СКВ-сервера или качать с него старую версию файла и делать локальное сравнение.

Перед сохранением любого файла Git вычисляет контрольную сумму, и она становится индексом этого файла. Поэтому невозможно изменить содержимое файла или каталога так, чтобы Git не узнал об этом. Эта функциональность встроена в сам фундамент Git'а и является важной составляющей его философии. Если информация потеряется при передаче или повредится на диске, Git всегда это выявит.

Практически все действия, которые вы совершаете в Git'е, только добавляют данные в базу. Очень сложно заставить систему удалить данные или сделать что-то неотменяемое. Можно, как и в любой другой СКВ, потерять данные, которые вы ещё не сохранили, но как только они зафиксированы, их очень сложно потерять, особенно если вы регулярно отправляете изменения в другой репозиторий.

# Основные задачи работы:

1. Создать учетную запись на github.com

2. Создать локальный репозиторий

3. Зафиксировать изменения в области заготовленных файлов

4. Переслать локальный коммит на сервер

5. Сделать слияние веток

6. Просмотреть изменения и разрешение конфликтов

7. Удалить ветку на сервере и вернуть к предыдущему состоянию

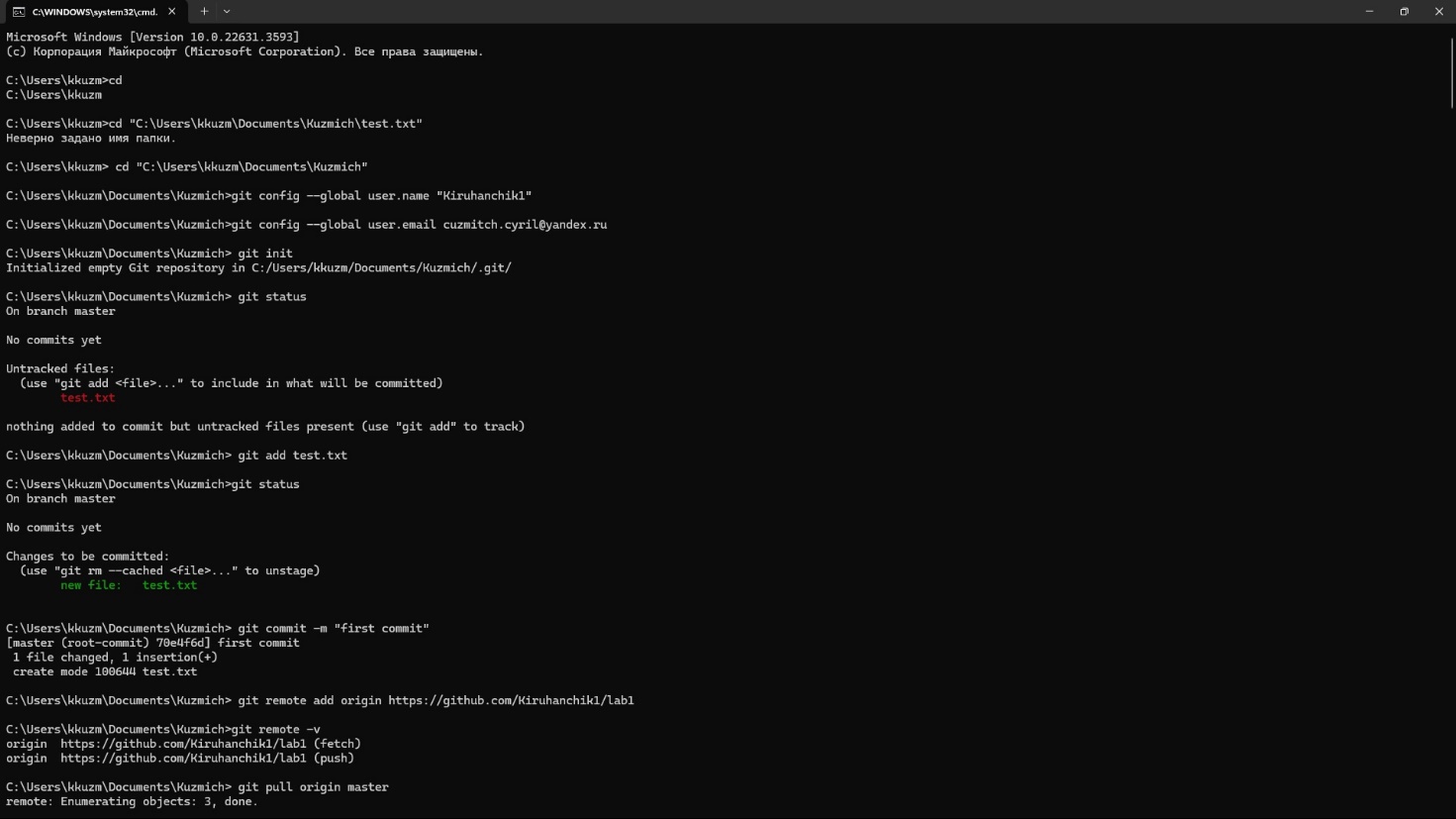
8. Исправить коммит.

# Ход работы:

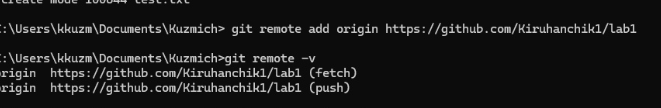
1. Создаем учетную запись на github.com или заходим в уже имеющийся личный кабинет, как в нашем случае и создаем новый репозиторий.

2. Создали файлик test.txt и написал наши фамилии. Открыли командную строку «cmd» и перешли с помощью команды «cd» в созданный нами каталог. Настроим самые важные опции и параметры: наше пользователя и адрес электронной почты.

3. Переходим в созданную нами папку, используя команду cd. Теперь необходимо проинициализировать эту папку как git репозиторий. Вводим команду: git init. Командная строка возвращает нам сообщение, что проинициализирована пустая git директория. Теперь используя проводник windows создаем в своей папке текстовый файл и пишем в него наши имена. Вводим команду: git stаtus и нам отображается сообщение, что есть новый файл, но он не отслеживается. Чтобы исправить ситуацию используем команды add . и commit.

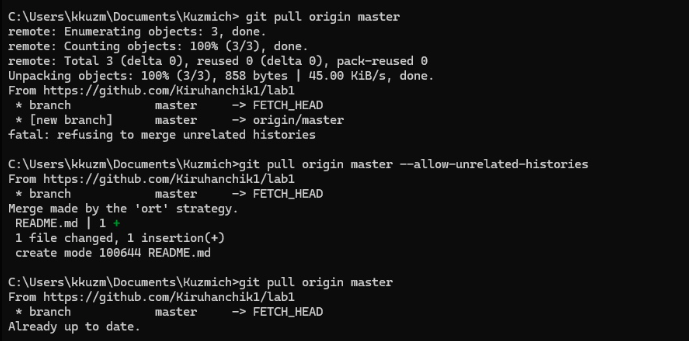


4. Чтобы связать наш локальный репозиторий с репозиторием на GitHub, выполним следующую команду в терминале

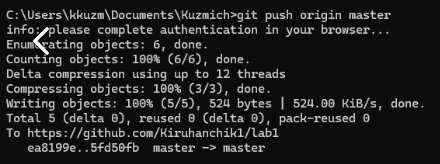


5. Запросим изменения с сервера

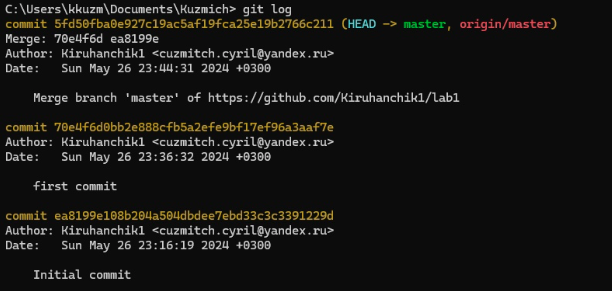
Мы создали изменения в нашем репозитории, создав файл readme и другие пользователи могут скачать изменения при помощи команды pull



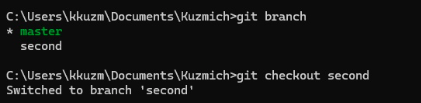
6. Теперь перешлем локальный коммит на сервер



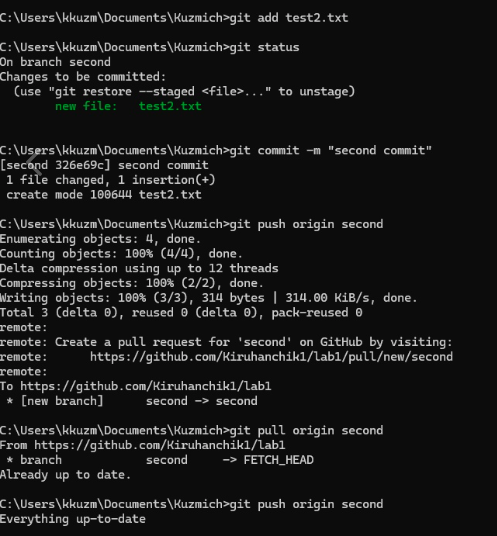
Посмотрим наши изменения с использованием команды



7. Создадим новую ветку second, посмотрим в какой находимся и перейдем на другую:

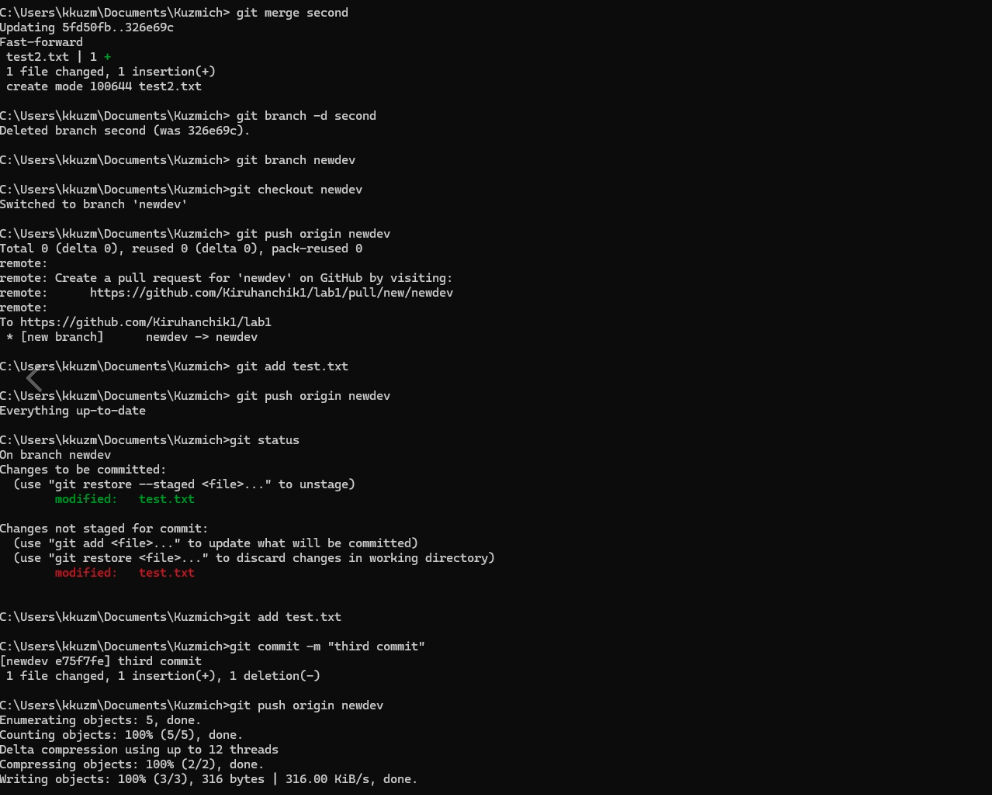


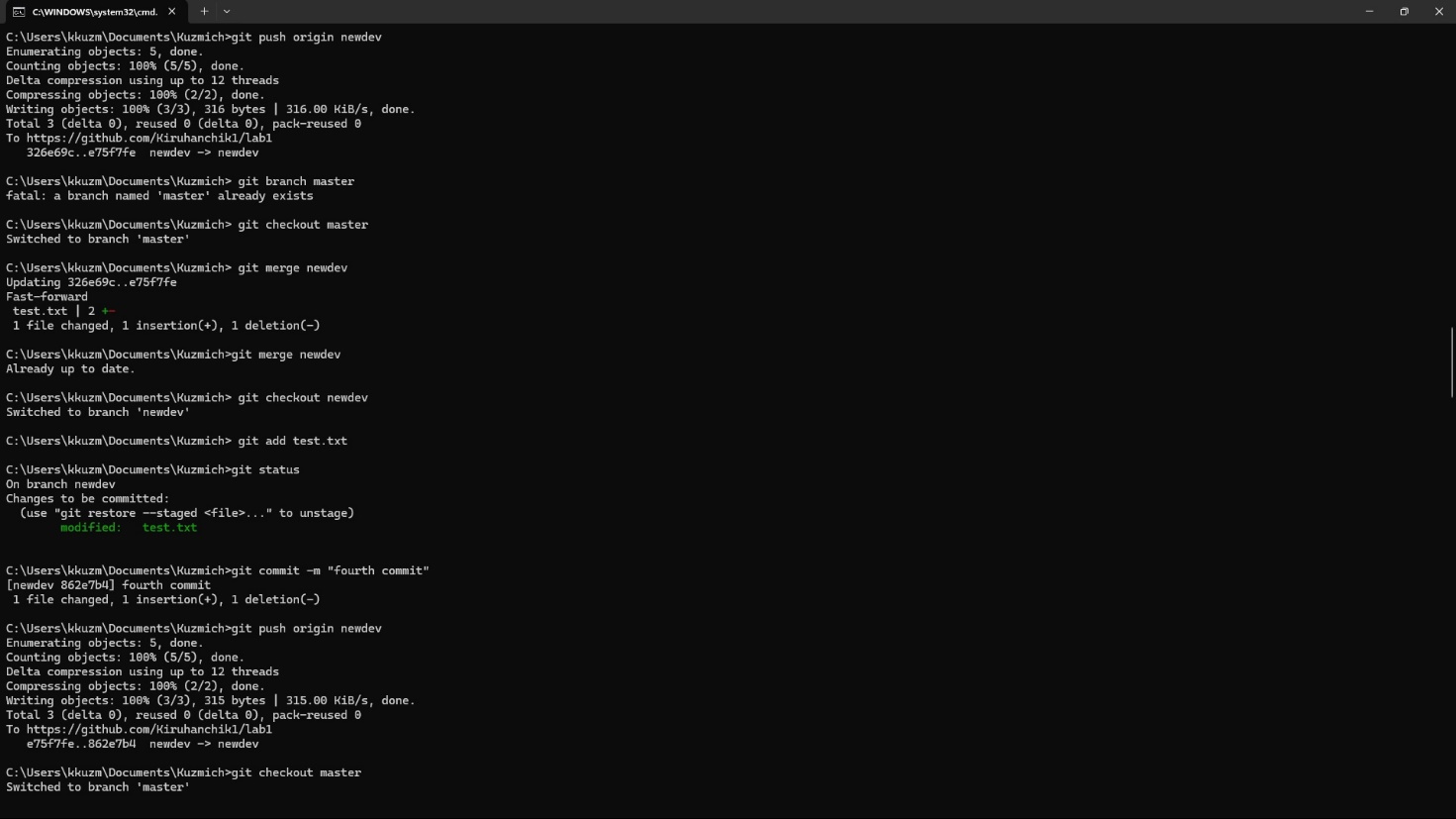
8. Создаем новый файл в нашем локальном репозитории и пишем в нем свои фамилии, добавляем в область подготовленных файлов, коммитим и отправляем на сервер: так же сливаем ветки и удаляем second



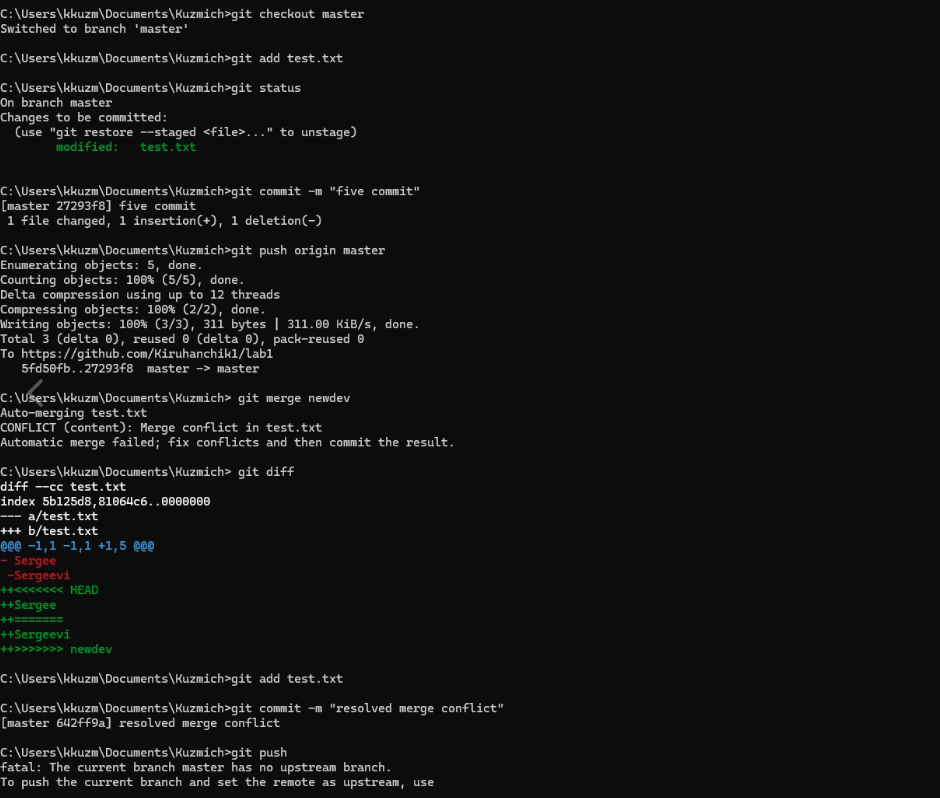
9. Слияние веток

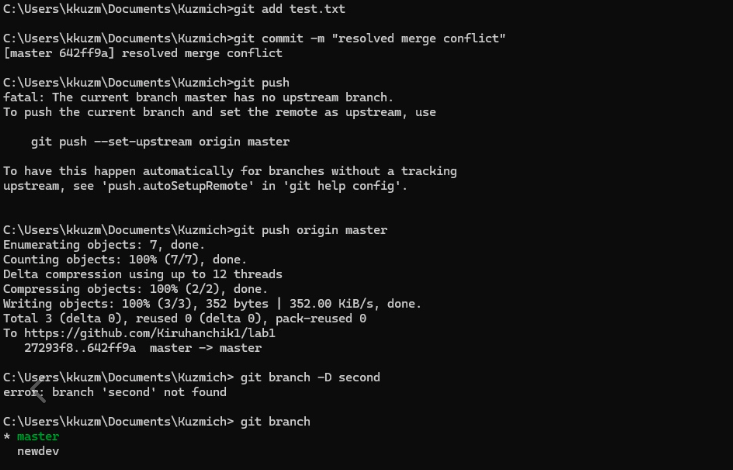






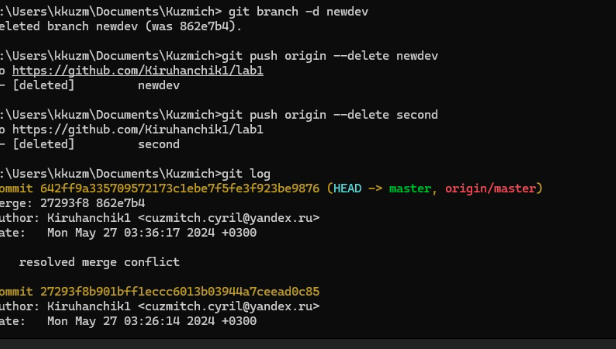
10. Просмотр изменений и разрешение конфликтов





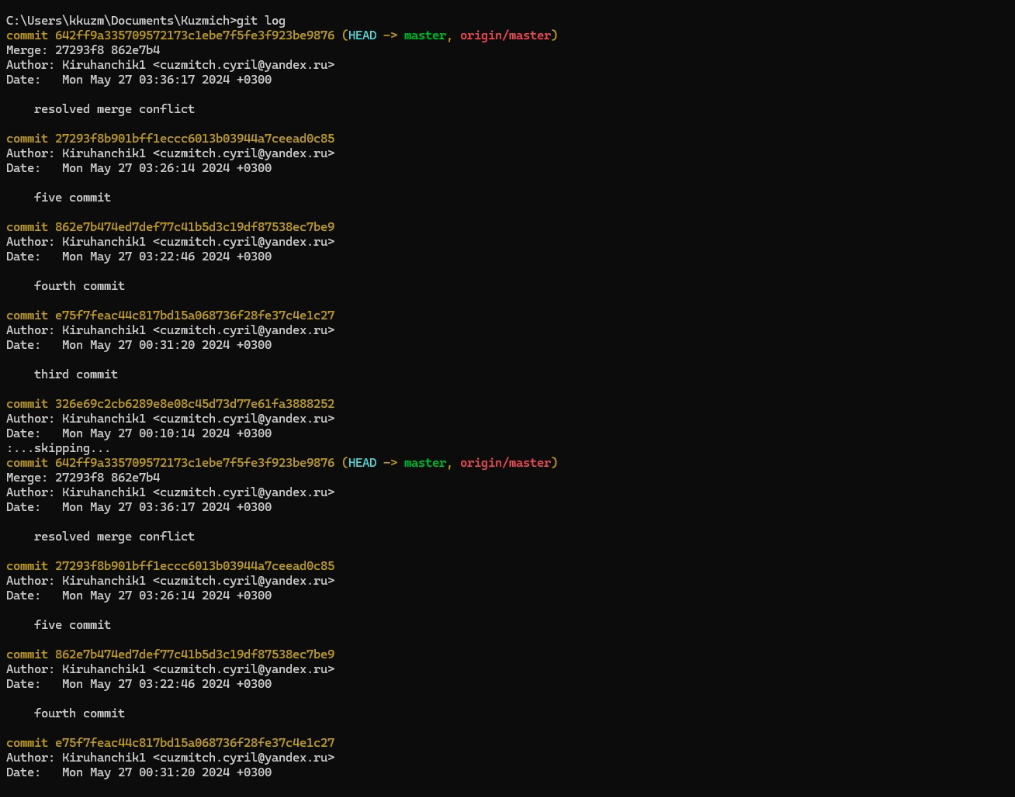
11.Удаление веток на сервере

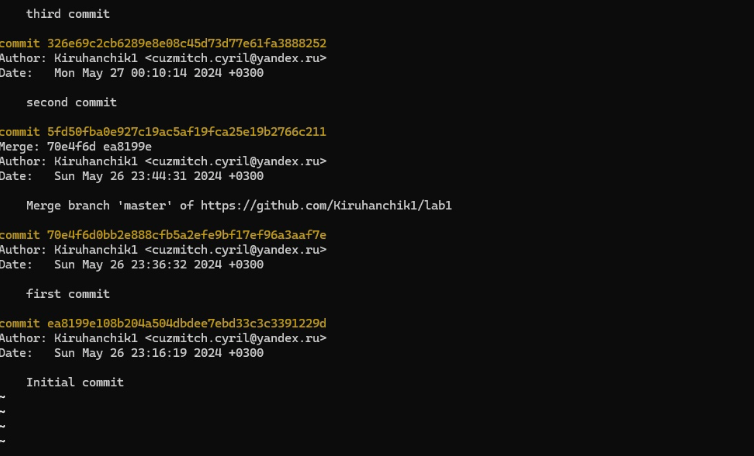
Удаляем ветки с github:



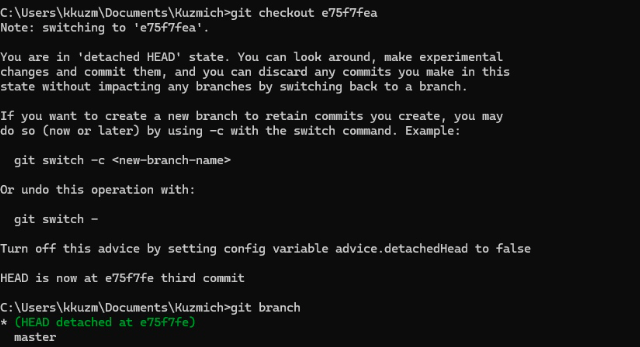
1. Возврат к предыдущему состоянию

Чтобы посмотреть все комиты, можно использовать команду git log:

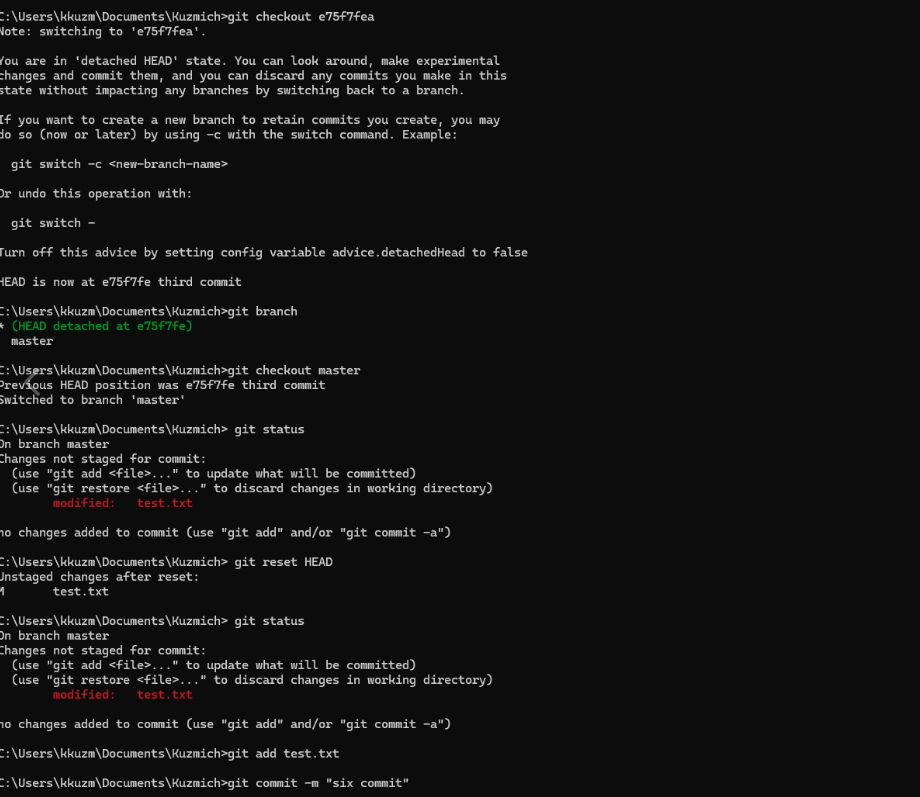


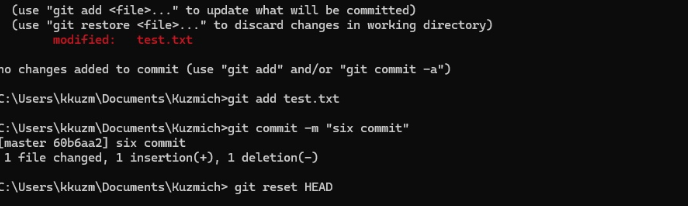


Выбираем коммит и откатываемся:



1. Исправление коммита





1. Отправка только нужных файлов на сервер

Создаем вручную файл под названием «.gitignore» и сохраняем его в

директорию проекта. Внутри файла перечисляем названия файлов/папок,

которые нужно игнорировать, каждый с новой строки. Файл «.gitignore»

добавляем, коммитим и отправляем на сервер.

# Вывод:

В лабораторной работе №1 *«Работа с системами контроля версий на примере GitHub»* мы освоили навыки работы с системами контроля версий на примере GitHub.

https://github.com/Kiruhanchik1/lab1

# Список использованных литературы:

1. «HTML, XHTML and CSS», Andy Harris. Изд.: Машиностроение, 2012.
2. «JavaScript», Дэвид Макфарланд. Изд.: Символ-Плюс, 2013.
3. «Большая книга CSS3», Дэвид Макфарланд. Изд.: Питер, 2016.
4. Интернет-ресурсы.