Лабораторная работа №1

Работа с git

Кудякова София Андреевна

Содержание

# 1 Цель работы

Цель данной лабораторной работы - приобретение практических навыков работы с системой управления версиями Git.

# 2 Задание

1. Выполнить задания из лабораторной работы
2. Оформить отчет по лабораторной работе с помощью Markdown

# 3 Теоретическое введение

Git — распределённая система управления версиями. Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux, первая версия выпущена 7 апреля 2005 года; координатор — Дзюн Хамано. [**wiki:bash?**].

# 4 Выполнение лабораторной работы

Использованы материыл из [**g?**].

Ранее я уже работа с git, поэтому установка у меня уже осуществлена. Перейдем сразу к параметрам установки окончаний строк.

## 4.1 Подготовка

Сначала настроим core.autocrlf с параметрами true и input, чтобы сделать все переводы строк текстовых файлов в главном репозитории одинаковыми, а затем настроим отображение unicode(рис. [[1](#fig:001)]).

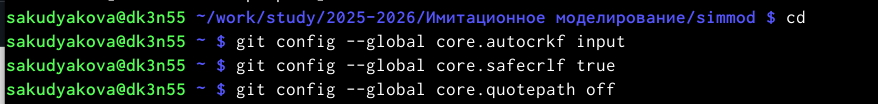


Figure 1: Настройка git

## 4.2 Создание проекта

Создадим пустой каталог hello, а в нём файл с именем hello.html. В файл добавим строчку “Hello, World!”. Затем создадим git репозиторий из этого каталога, выполнив команду git init. (рис. [[2](#fig:002)]).

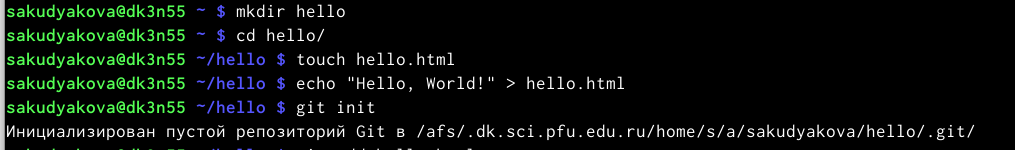


Figure 2: Создание репозитория

Добавим файл в репозиторий и проверим статус, мы увидим сообщение о том, что коммитить нечего (рис. [[3](#fig:003)]).

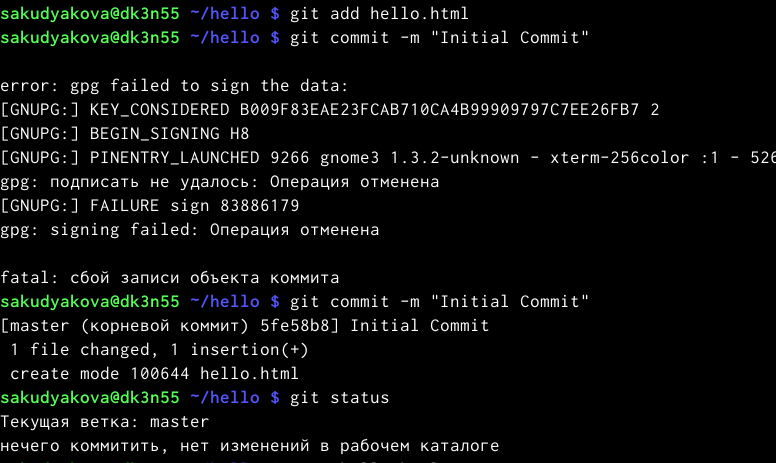


Figure 3: Добавление файла в репозиторий

## 4.3 Внесение изменений

Изменим содержимое файла hello.html на:

<h1>Hello, World!</h1>

Проверив состояние рабочего каталога увидим, что git знает, что файл hello.html был изменен, но при этом эти изменения еще не зафиксированы в репозитории. Проиндексируем изменения и снова посмотрим статус, в нем указано, что изменения пока не записаны в резапозиторий. И наконец закоммитим изменения, внеся их в репозиторий и снова посмотрим статус, который теперь показывает, что все изменения внесены в репозиторий (рис. [[4](#fig:004)]).

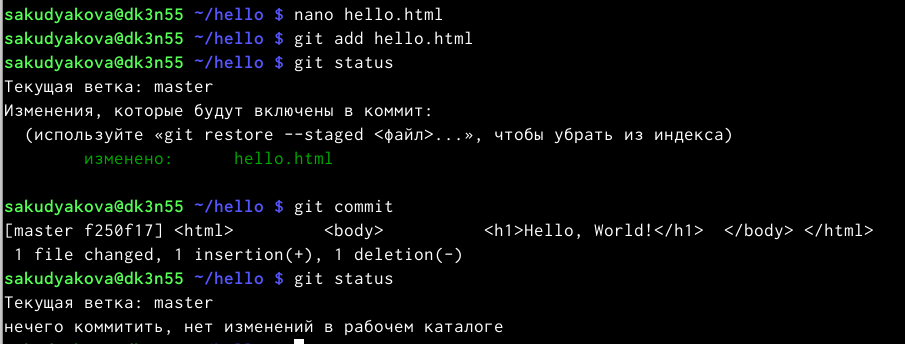


Figure 4: Внесение изменений в содержимое репозитория

Изменим страницу «Hello, World», чтобы она содержала стандартные теги

и

(рис. [[5](#fig:005)]).

<html>  
 <body>  
 <h1>Hello, World!</h1>  
 </body>  
</html>

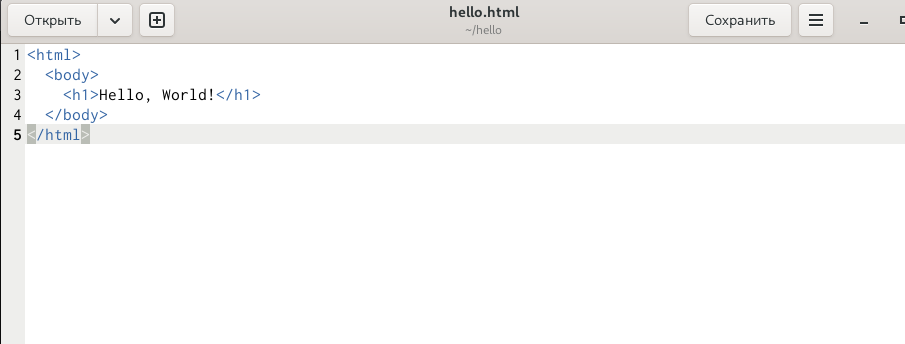


Figure 5: Изменение файла hello.html

Добавим это изменение в индекс git и добавим заголовки HTML (секцию

) к странице «Hello, World». (рис. [[6](#fig:006)])

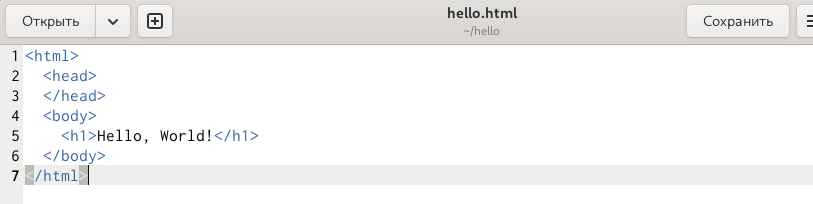


Figure 6: Изменение файла hello.html

Проверив текущий статус, увидим, что hello.html указан дважды в состоянии. Первое изменение (добавление стандартных тегов) проиндексировано и готово к коммиту. Второе изменение (добавление заголовков HTML) не проиндексированно. (рис. [[7](#fig:007)]).

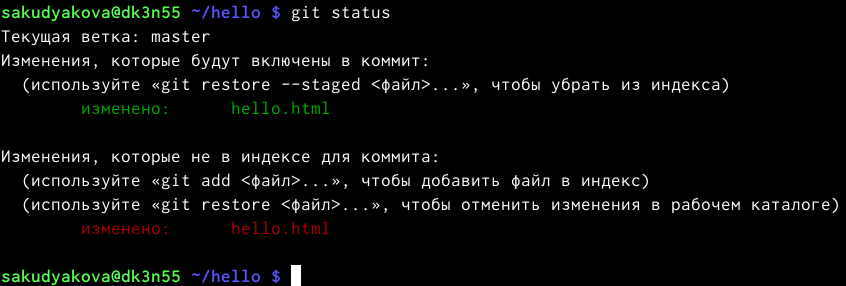


Figure 7: Внесение нескольких изменений в содержимое репозитория

Произведем коммит проиндексированного изменения. Проиндексируем оставшееся изменение, посмотрим статус и прокоммитим его тоже (рис. [[8](#fig:008)]).

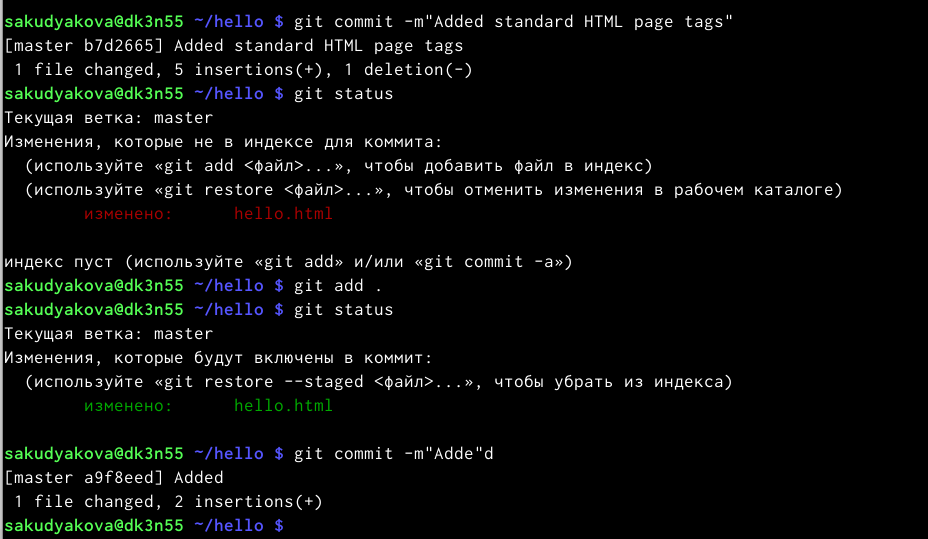


Figure 8: Внесение нескольких изменений в содержимое репозитория

## 4.4 История

Получим список произведенных изменений в стандартном виде git log, затем в однострочном git log --pretty=oneline и далее с указанием времени и количества (рис. [[10](#fig:009)]).

git log --pretty=oneline --max-count=2  
git log --pretty=oneline --since='5 minutes ago'  
git log --pretty=oneline --until='5 minutes ago'

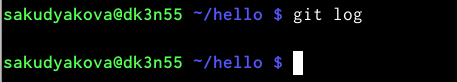
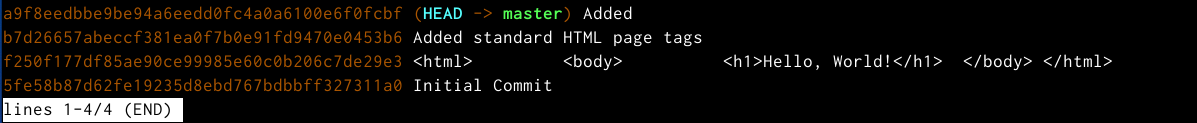
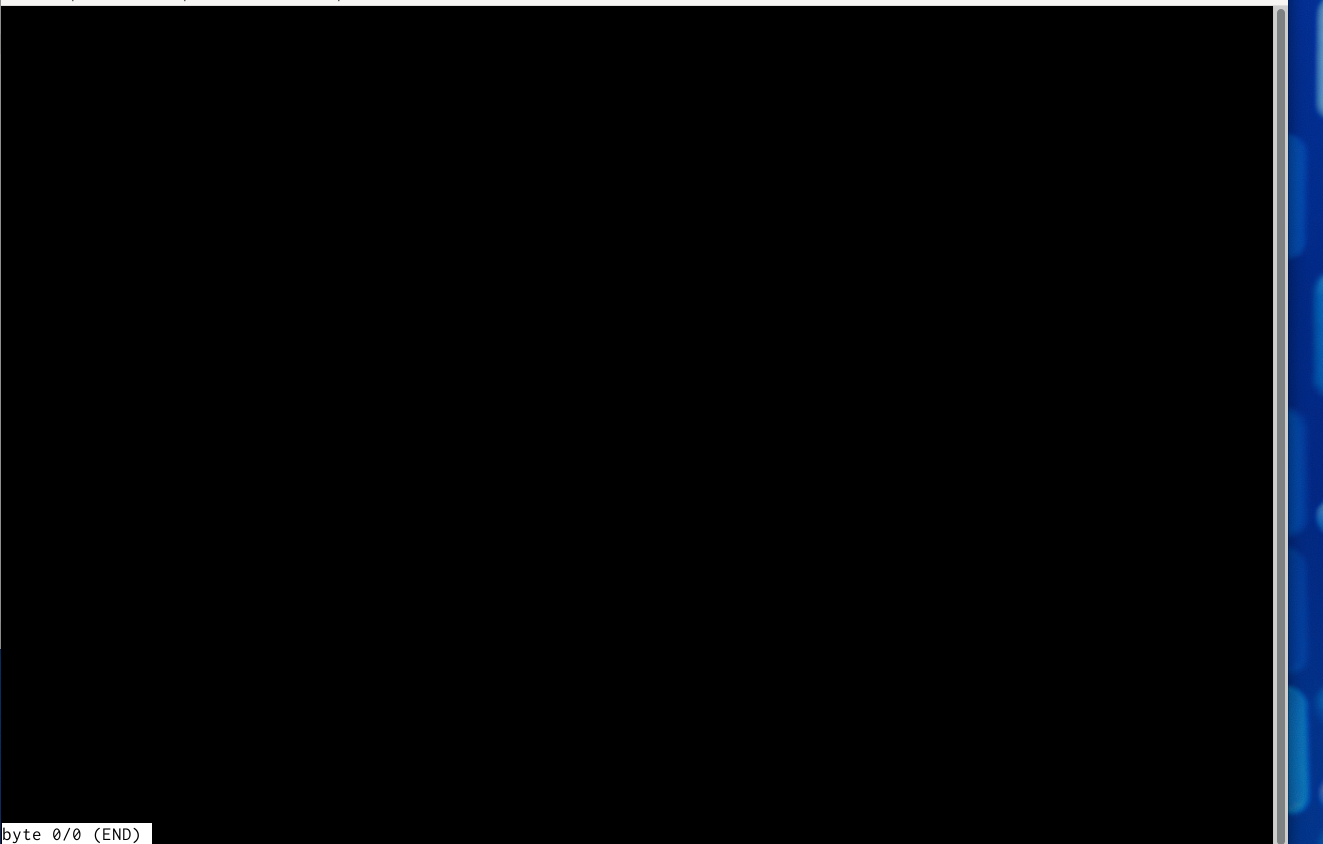
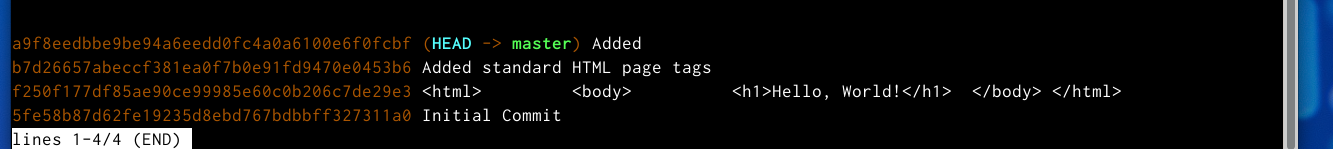


Figure 9: Просмотр истории

## 4.5 Получение старых версий

Изучим данные лога и найдем там хэш первого коммита, используя его, вернемся к первой верссии репозитория и просмотрим файл hello.html, действительно, увидим первую версию. Затем вернемся к последней версии в ветке master и посмотрим на тот же файл (рис. [[8](#fig:008)]).

 Просмотр разных версий репозитория  Просмотр разных версий репозитория Просмотр разных версий репозитория Просмотр разных версий репозитория  

## 4.6 Создание тегов версий

Назовем текущую версию страницы hello первой (v1). Создадим тег первой версии git tag v1 и используем его для того, чтобы вернуться к предыдущей версии, которой также присвоим тег v1-beta (рис. [[10](#fig:009)]).

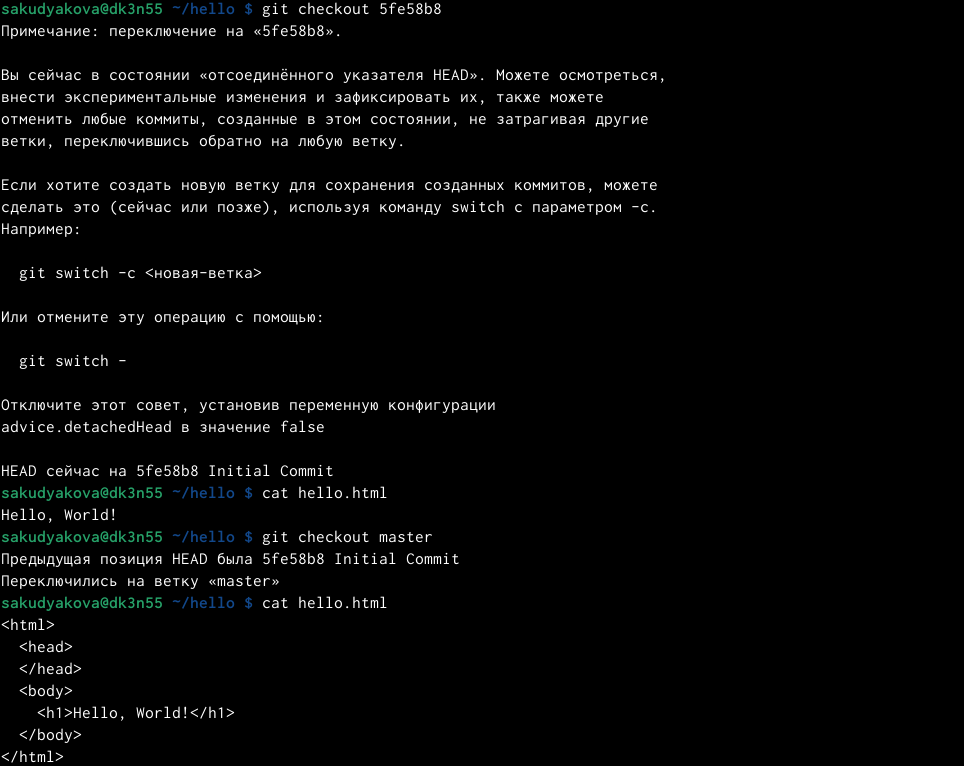


Figure 10: Создание тегов версий

Переключимся по тегам между двумя отмеченными версиями. Просмотрим все доступные теги (их два) и посмотрим теги в логе (рис. [[11](#fig:010)]).

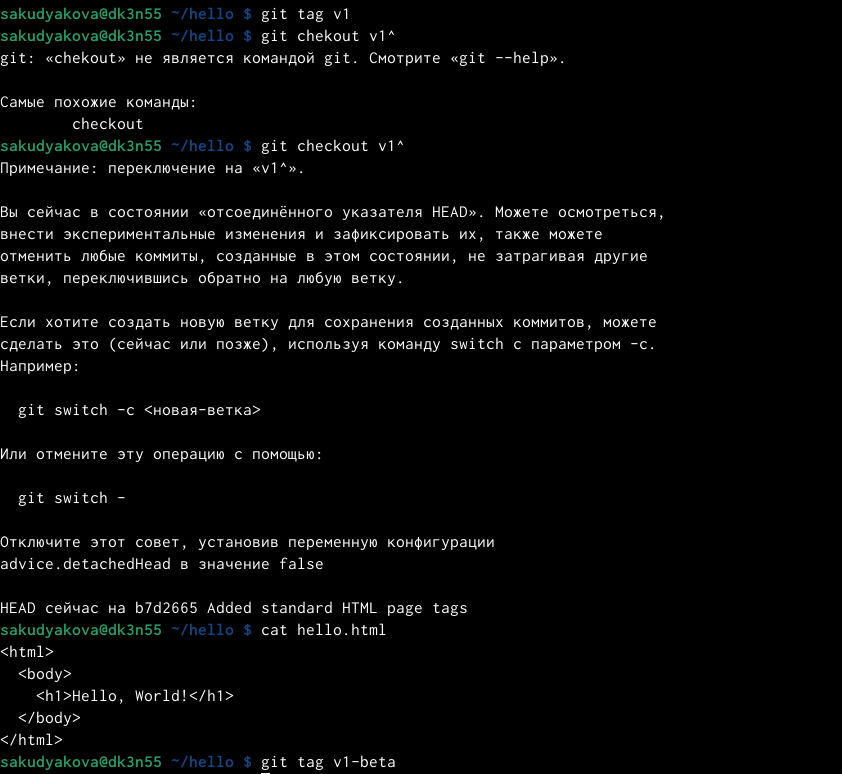
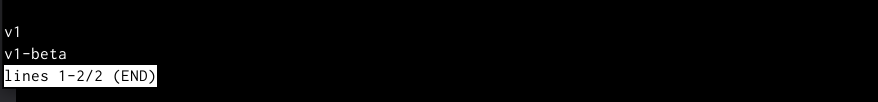
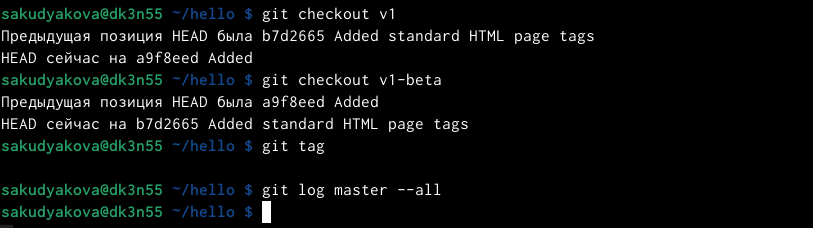


Figure 11: Переключение по имени тега и просмотр доступных тегов

## 4.7 Отмена локальных изменений (до индексации)

Убдеимся, что мы находимся на последнем коммите ветки master и внесем изменение в файл hello.html в виде нежелательного комментария (рис. [??]).

Затем проверим статус, увидим, что изменения ещё не проиндексированы. Используем команду git checkout для переключения версии файла hello.html в репозитории, видим, что нежелательный комментарий отсутствует в файле (рис. [[12](#fig:011)]).



Figure 12: Отмена локальных изменений (до индексации)

## 4.8 Отмена проиндексированных изменений (перед коммитом)

Внесем изменение в файл hello.html в виде нежелательного комментария

<html>  
 <head>  
 <!-- This is an unwanted but staged comment -->  
 </head>  
 <body>  
 <h1>Hello, World!</h1>  
 </body>  
</html>

Проиндексируем это изменение и проверим состояние. Состояние показывает, что изменение было проиндексировано и готово к коммиту. Используем команду git reset, чтобы сбросить буферную зону к HEAD. Это очищает буферную зону от изменений, которые мы только что проиндексировали. Переключимся на последнюю версию коммита, посмотрев статус увидим, что наш каталог опять чист (рис. [[13](#fig:012)]).

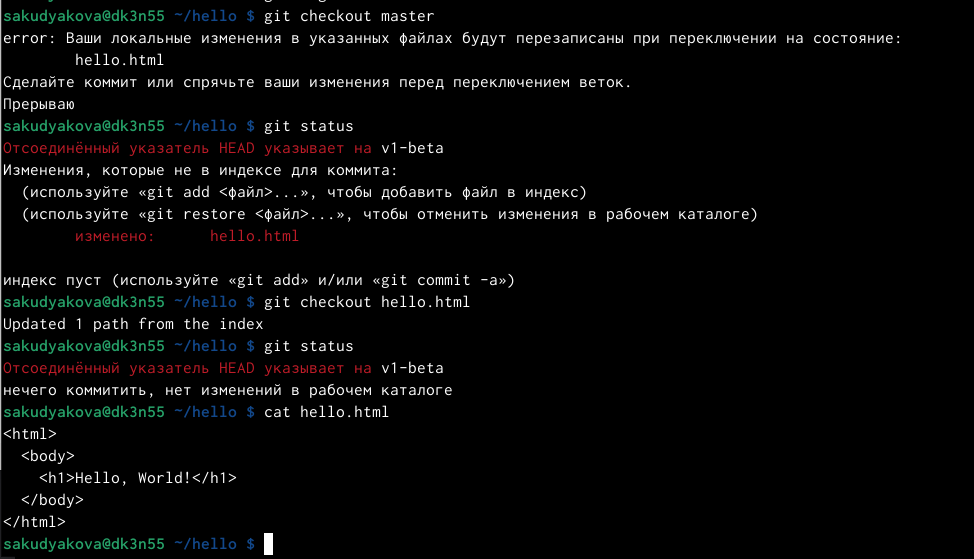


Figure 13: Отмена проиндексированных изменений (перед коммитом)

## 4.9 Отмена коммитов

Изменим файл hello.html на следующий.

<html>  
 <head>  
 </head>  
 <body>  
 <h1>Hello, World!</h1>  
 <!-- This is an unwanted but committed change -->  
 </body>  
</html>

Проиндексируем изменения файла и прокоммитим их. Чтобы отменить коммит, нам необходимо сделать коммит, который удаляет изменения, сохраненные нежелательным коммитом. Перейдем в редактор, где изменим нежелательный коммит. Проверим лог. Проверка лога показывает нежелательные и отмененные коммиты в нашем репозитории (рис. [[14](#fig:013)]).

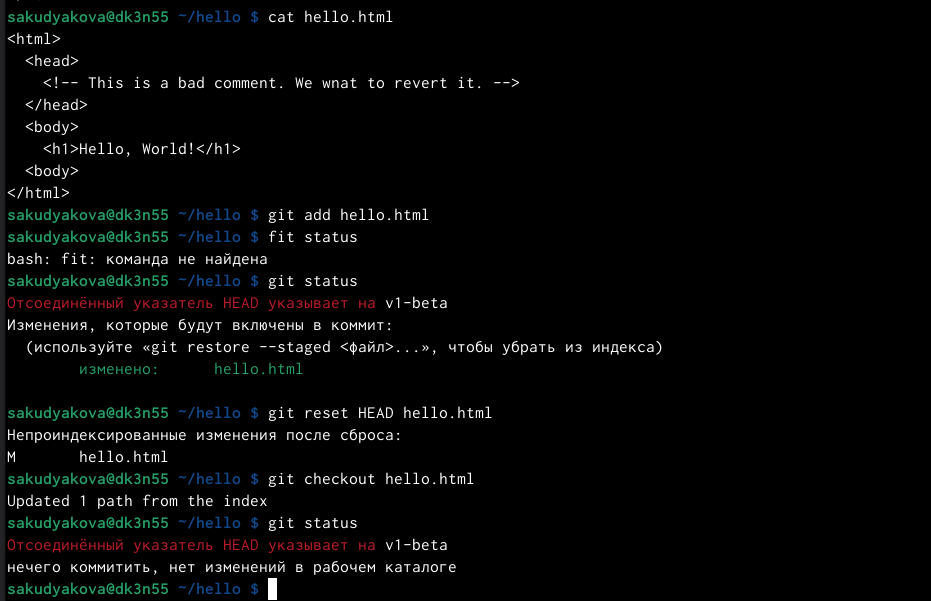


Figure 14: Отмена коммитов

## 4.10 Удаление коммиттов из ветки

Удалим последние два коммита с помощью сброса, сначала отметим последний коммит тегом, чтобы его можно было потом найти. Используем командуgit reser --hard v1, чтобы вернуться к версии до этих коммитов. (рис. [[15](#fig:014)]).

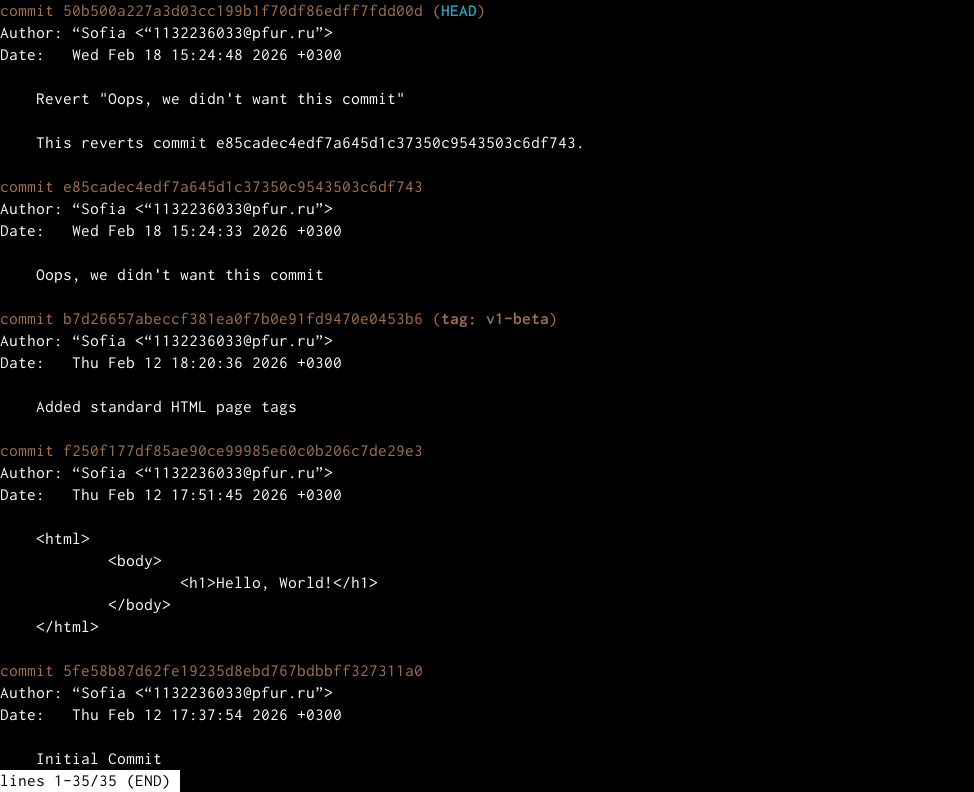


Figure 15: Удаление коммиттов из ветки

Теперь в логе их нет, но если посмотреть логи с опцией --all можно их увидеть, однако метка HEAD находится на нужной нам версии (рис. [[16](#fig:015)]).

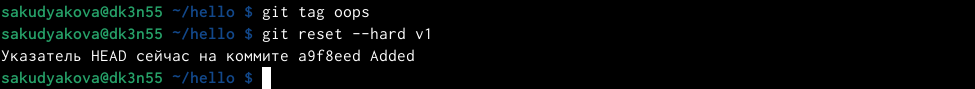


Figure 16: Ничего никогда не теряется

## 4.11 Удаление тега oops

Удалим тег oops и коммиты, на которые он ссылался, сборщиком мусора. Теперь этот тег не отображается в репозитории (рис. [[17](#fig:016)]).

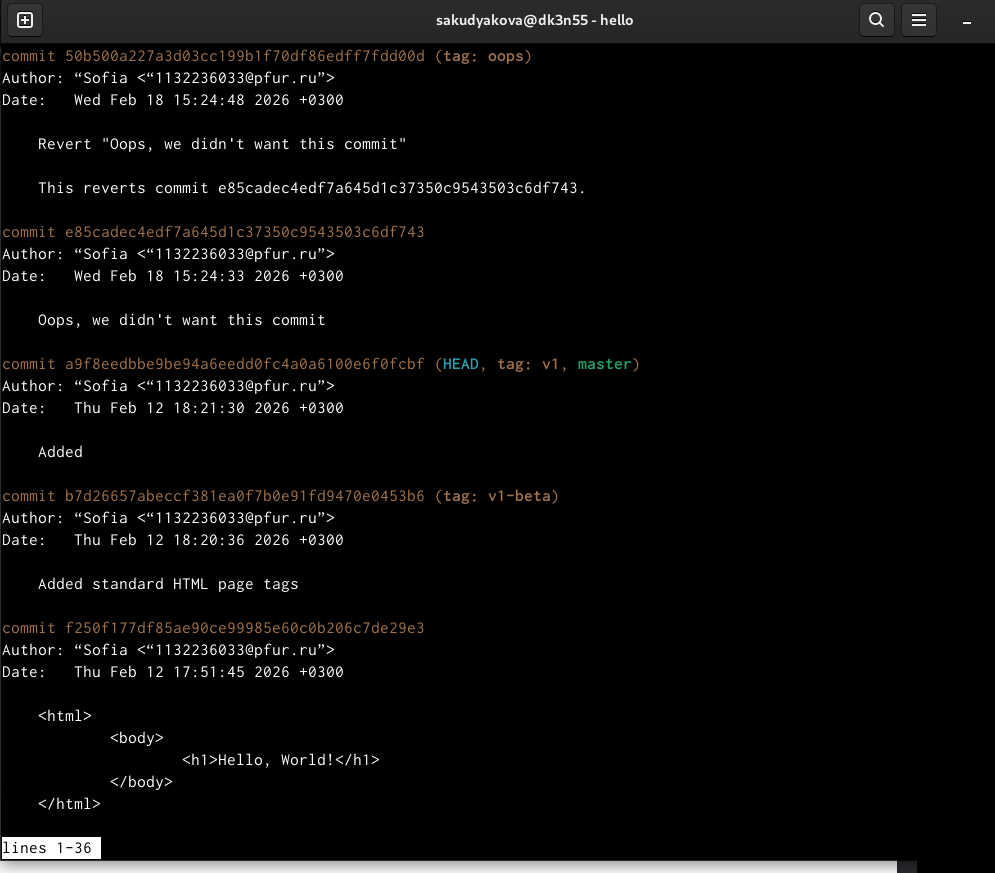


Figure 17: Удаление тега oops

## 4.12 Изменение предыдущего коммита

Добавим в страницу комментарий автора (рис. [[18](#fig:017)]).



Figure 18: Изменение файла hello.html

Затем добавим изменения в репозиторий. Теперь мы хотим добавить в комментарий автора почту, обновим страницу hello, включив в неё почту. Чтобы у нас остался один коммит, а не два, изменим последний с помощь опции --amend, теперь в логах отображается последняя версия коммита (рис. [[19](#fig:018)]).

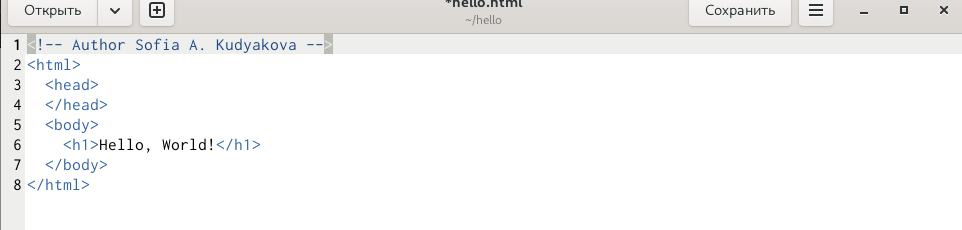
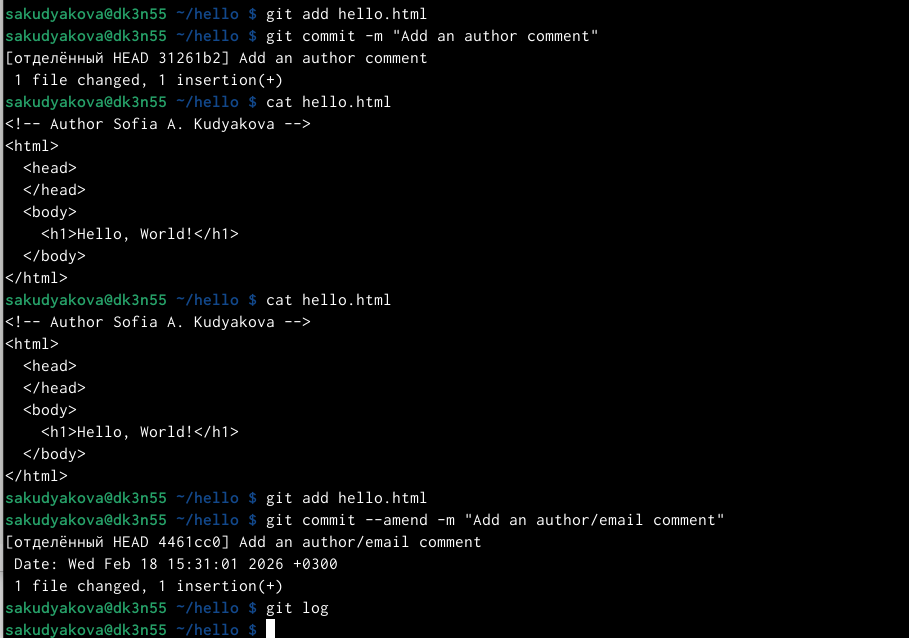
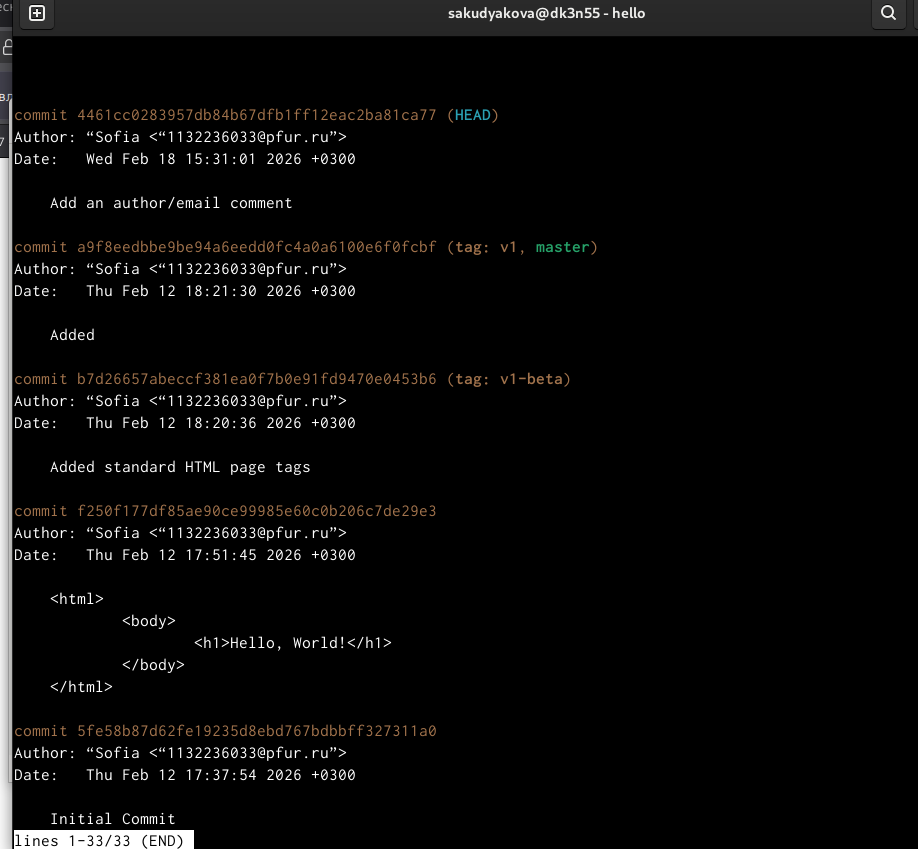
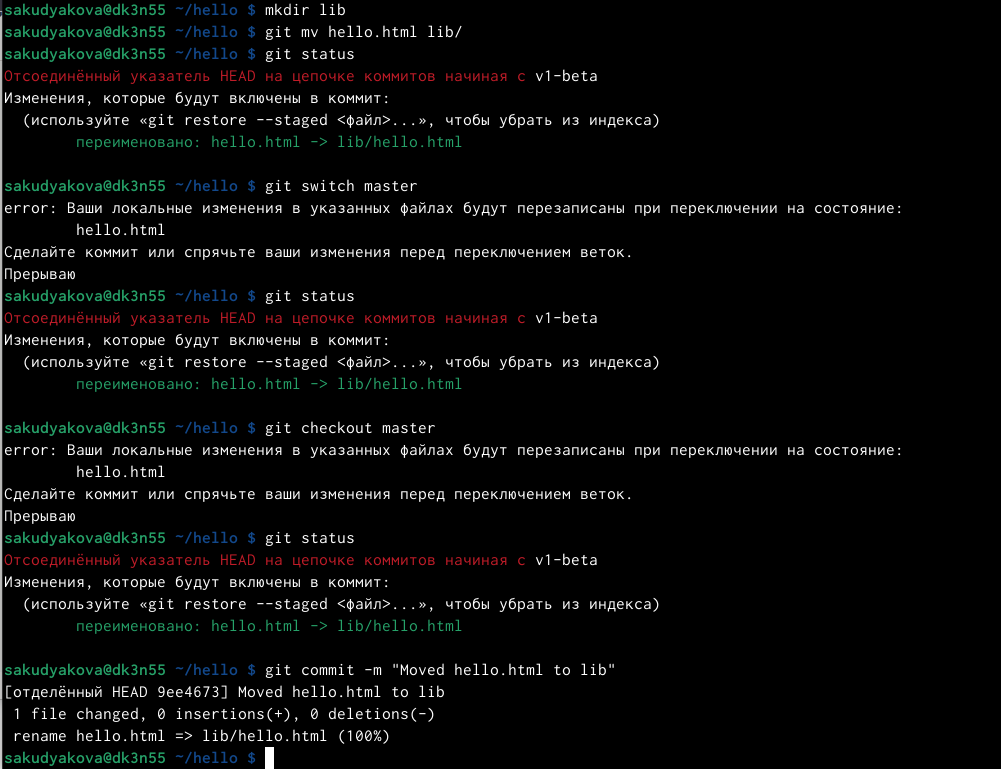


Figure 19: Изменение предыдущего коммита

## 4.13 Перемещение файлов

Переместим наш файл в каталог lib. Для этого создадим его и используем команду git mv, сделаем коммит этого перемещения (рис. [??]).

## 4.14 Подробнее о структуре

Добавим файл index.html в наш репозиторий, заполним файл:

<html>  
 <body>  
 <iframe src="lib/hello.html" width="200" height="200" />  
 </body>  
</html>

Проиндексируем файл и сделаем коммит (рис. [[20](#fig:0182)]).

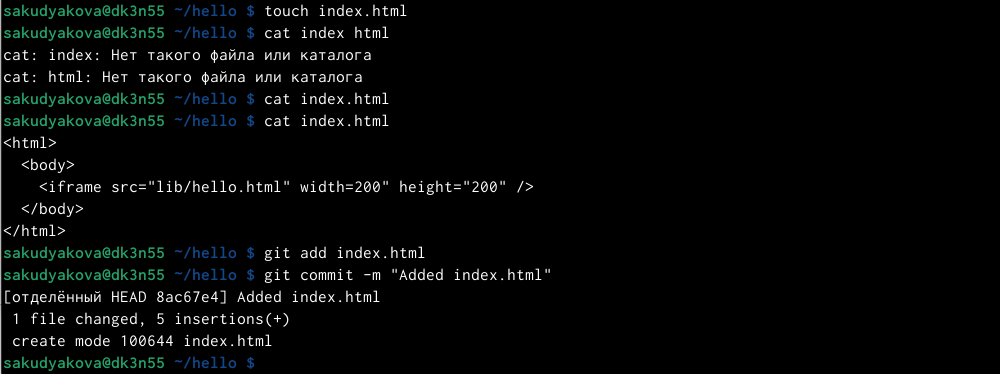


Figure 20: Создание, индексация, коммит нового файла

Теперь при открытии index.html увидим кусок страницы hello в маленьком окошке (рис. [[21](#fig:019)]).

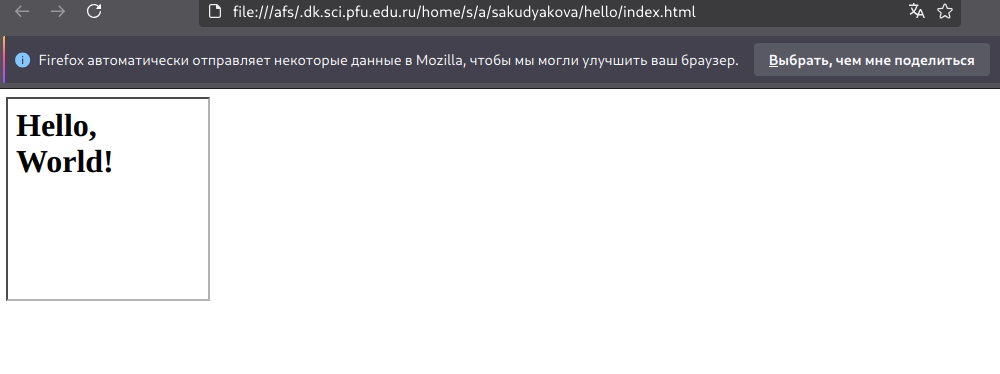


Figure 21: Результат открытия index.html

## 4.15 Git внутри: Каталог .git

Просмотрим каталог, в котором хранится вся информация git. ls -C .git/

Затем посмотрим набор каталогов, имена которых состоят из 2 символов ls -C .git/objects

Имена каталогов являются первыми двумя буквами хэша sha1 объекта, хранящегося в git. Посмотрим в один из каталогов с именем из 2 букв. ls -C .git/objects/ac Увидим файлы с именами из 38 символов. Это файлы, содержащие объекты, хранящиеся в git.

Посмотрим файл конфигурации, создающийся для каждого конкретного проекта cat .git/config. Затем посмотрим подкаталоги .git/refs/heads и .git/refs/tags, а также содержимое файла v1, в нём хранится хэш коммита, привязанный к тегу. Также посмотрим содержимоей файла HEAD, который содержит ссылку на текущую ветку, в данный момент это ветка master (рис. [[22](#fig:020)]).

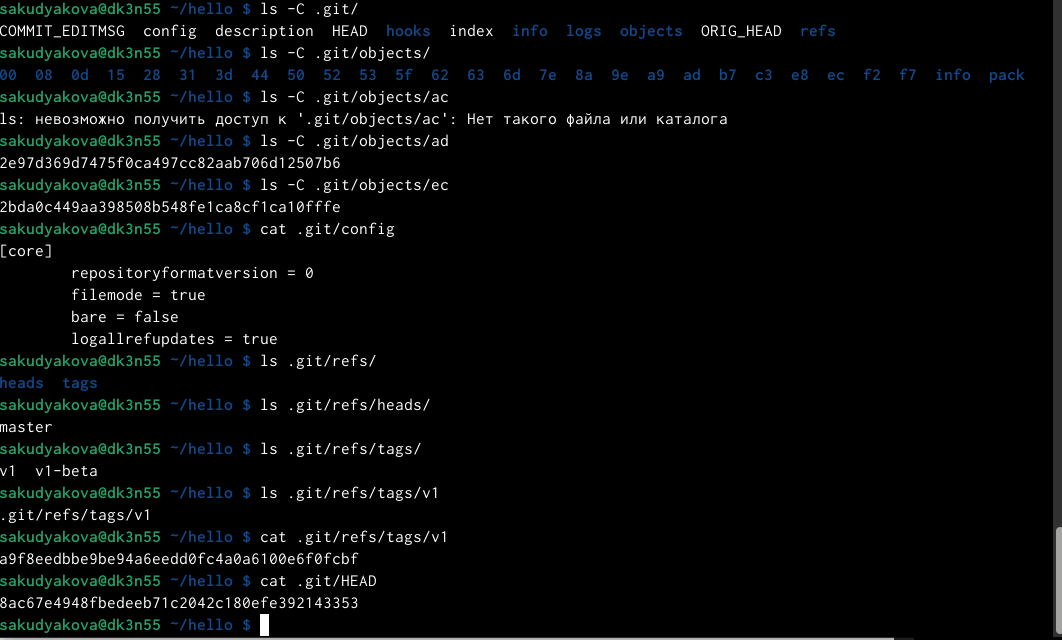
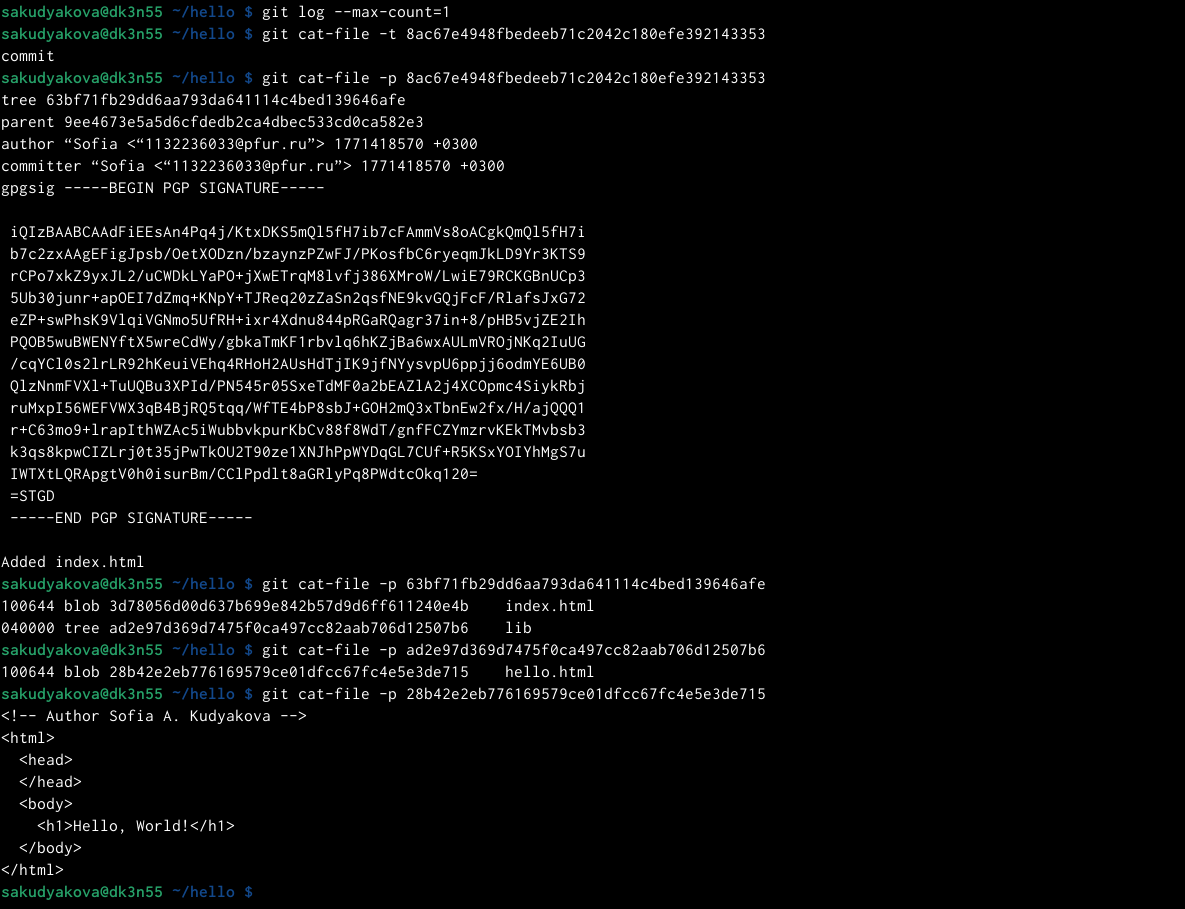
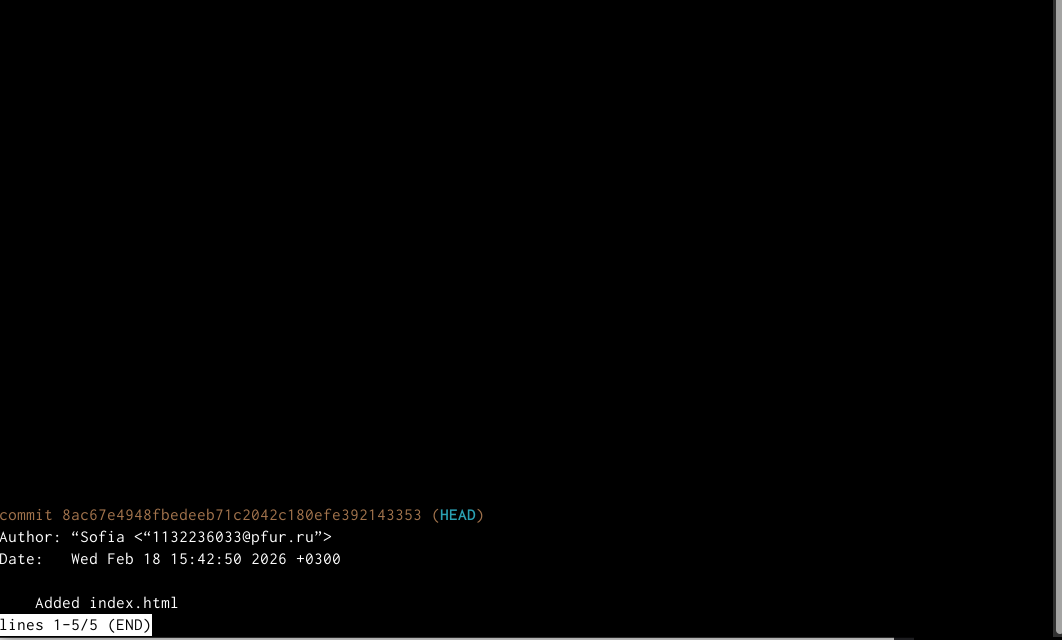


Figure 22: Каталог .git

## 4.16 Работа непосредственно с объектами git

Найдем последний коммит через log и выедем его, используя SHA1 хэш. Затем посмотрим дерево каталогов, ссылка на который идёт в последнем коммите, вывдем каталог lib и файл hello.html (рис. [??]).

Исследуем git репозиторий вручную самостоятельно. Используя хэш родительского коммита последовательно дойдем до первой версии файла hello.html и посмотрим его(рис. [[23](#fig:0211)]).

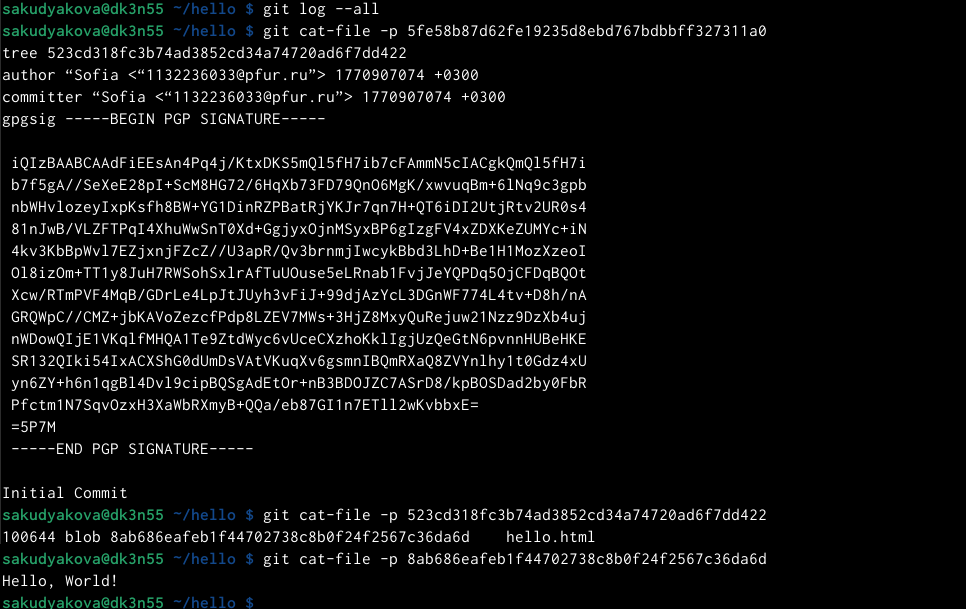


Figure 23: Самостоятельная работа непосредственно с объектами git

## 4.17 Создание ветки

Создадим новую ветку «style» и перейдем в неё. Добавим туда файл стилей style.css и добавим его в репозиторий (рис. [[24](#fig:022)]).

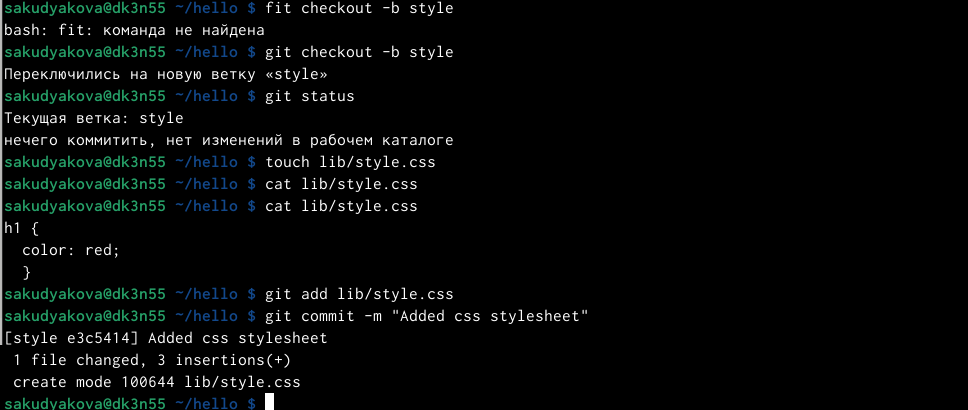


Figure 24: Создание ветки

Обновим файл hello.html, чтобы использовать стили style.css и index.html (рис. [[25](#fig:024)]).

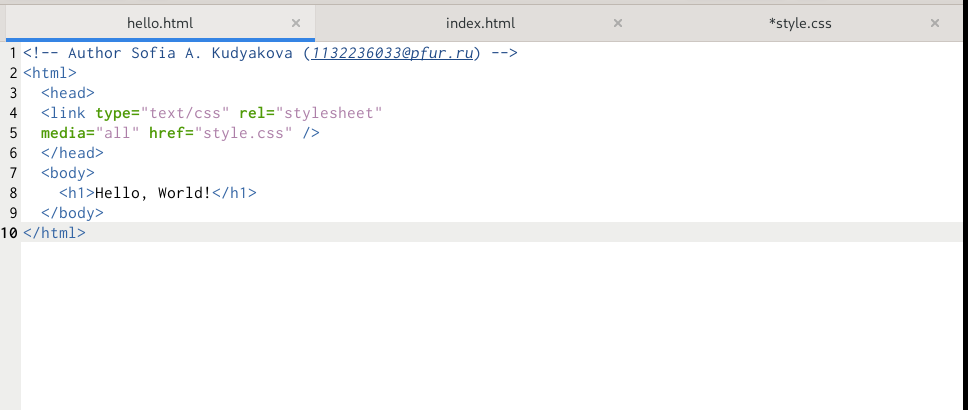


Figure 25: Редактирование файла hello.html

Обновиv файл index.html (рис. [[26](#fig:025)]).

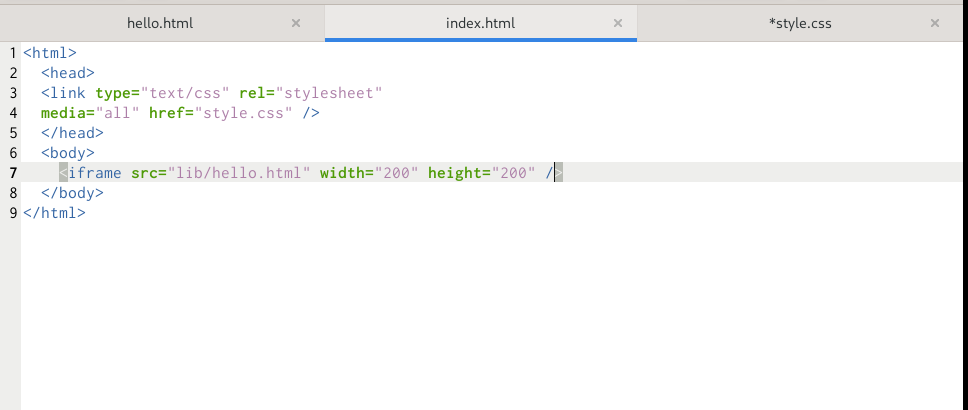


Figure 26: Редактирование файла index.html

Также обновим их в репозиторий (рис. [[27](#fig:023)]).

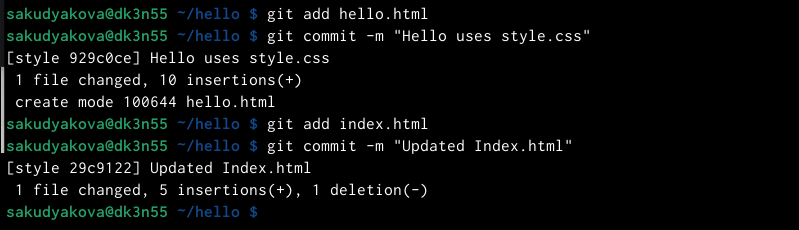


Figure 27: Создание ветки

## 4.18 Навигация по веткам

Посмотрим все логи (рис. [[28](#fig:026)]).

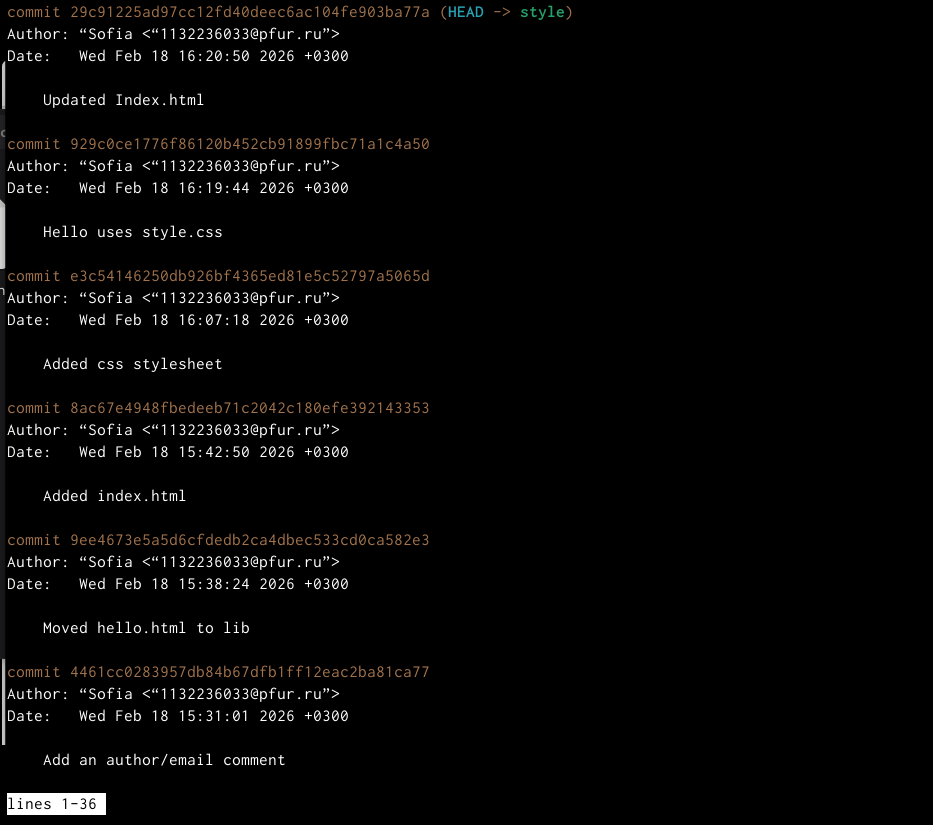
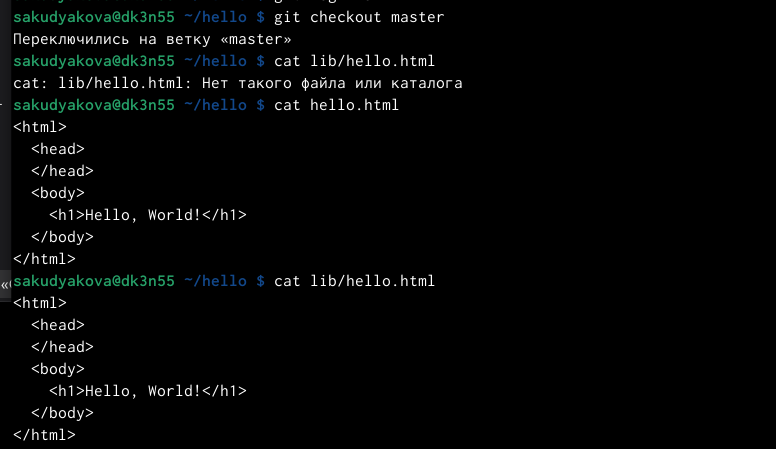
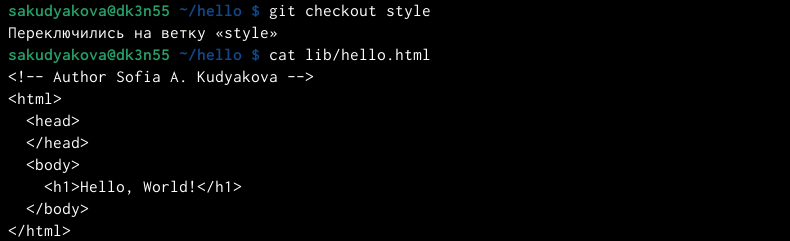


Figure 28: Просмотр логов новой ветки

Переключимся обратно на основную ветку и просмотрим содержимое файла lib/hello.html, заметим, что он не использует стили, не изменился, также просмотрим содержимое этого файла в новой ветке style, файл в ней уже использует стили(рис. [??]).

## 4.19 Изменения в ветке master

Вернемся в основную ветку и добавим файл README.md (рис. [[29](#fig:028)]).

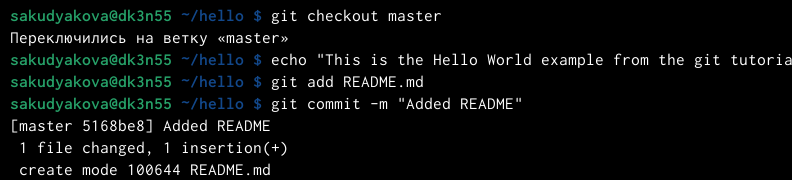


Figure 29: Изменения в ветке master

Просмотрим ветки и их различия git log --graph --all (рис. [[30](#fig:029)]).

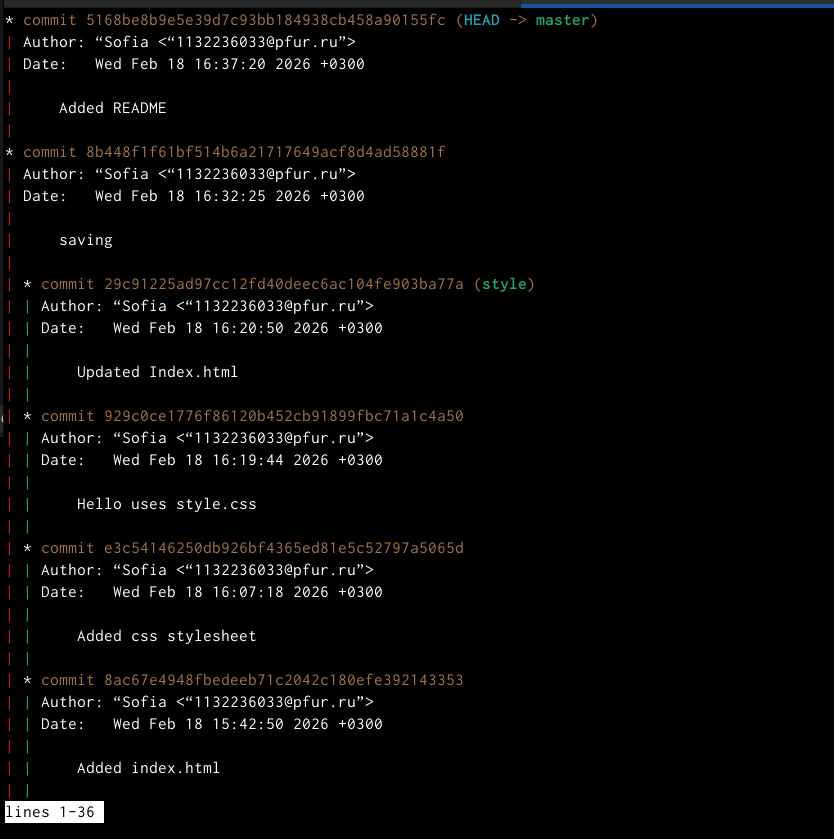
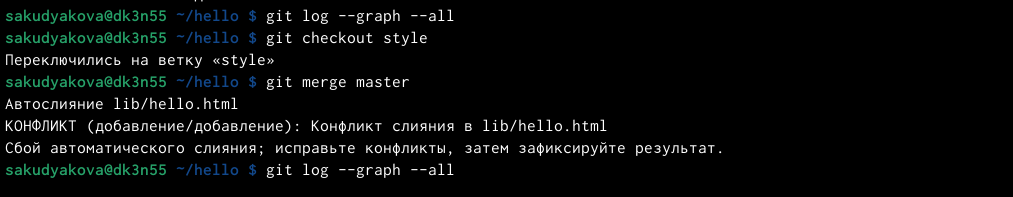
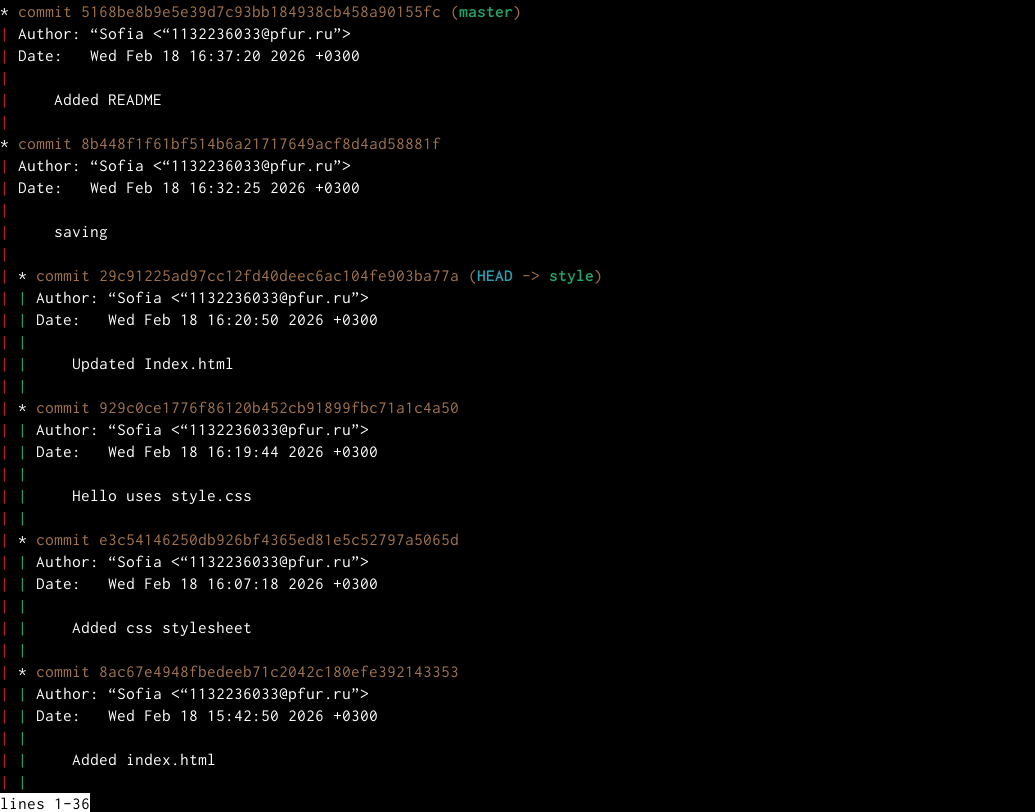


Figure 30: Изменения в ветке master

## 4.20 Слияние

Слияние переносит изменения из двух веток в одну. Вернемся к ветке style и сольем master с style (рис. [??]).

## 4.21 Создание конфликта

Вернемся в ветку master и создадим конфликт, внеся изменения в файл hello.html (рис. [[31](#fig:0301)]).

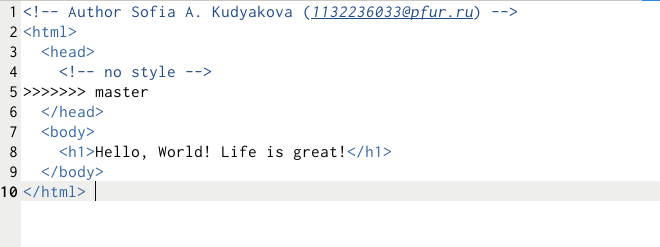
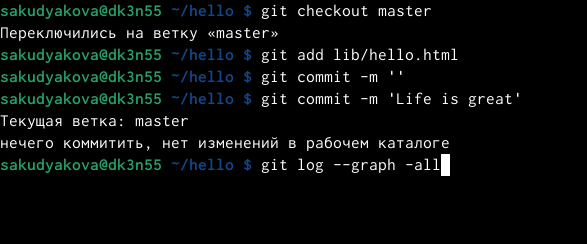
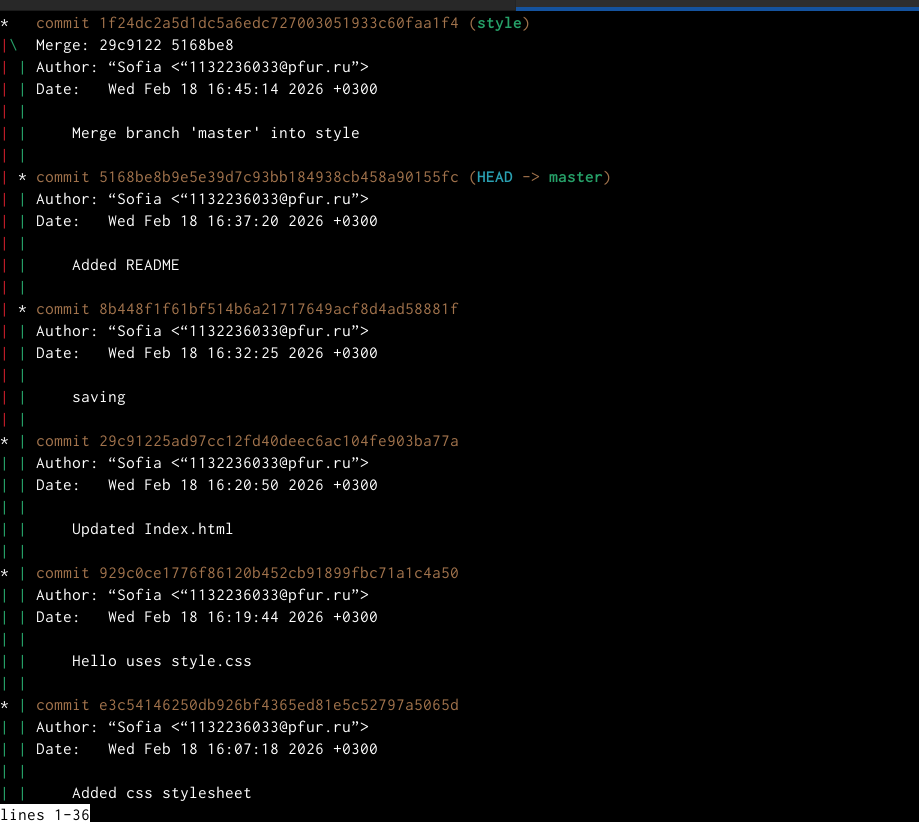


Figure 31: Создание конфликта

Просмотрим ветки. После коммита «Added README» ветка master была объединена с веткой style, но в настоящее время в master есть дополнительный коммит, который не был слит с style. Последнее изменение в master конфликтует с некоторыми изменениями в style (рис. [??]).

## 4.22 Разрешение конфликтов

Вернемся к ветке style и попытаемся объединить ее с новой веткой master. В файле lib/hello.html можно увидеть записи с обеих версий этого файла. Первый раздел — версия текущей ветки (style). Второй раздел — версия ветки master. Внесем изменения в lib/hello.html, оставив только необходимую нам запись и добавим этот файл в репозиторий, чтобы вручную разрешить конфликт (рис. [[32](#fig:032)],[[33](#fig:033)], [[34](#fig:034)]).

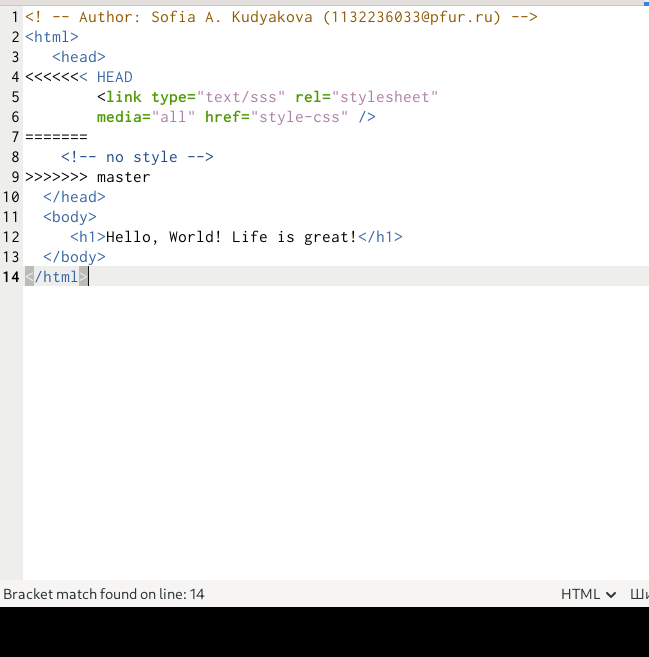


Figure 32: Конфликт



Figure 33: Разрешение конфликта

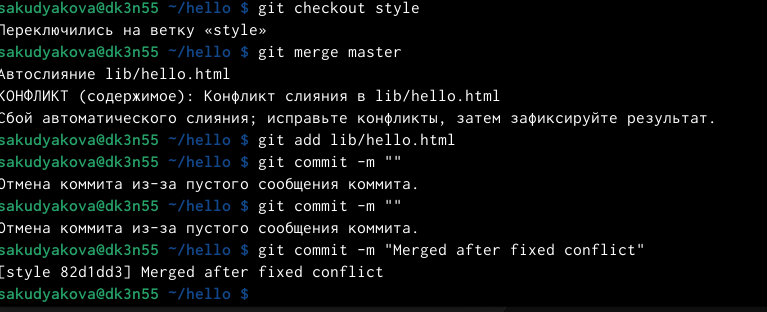


Figure 34: Коммит разрешения конфликта

## 4.23 Сброс ветки style

Вернемся на ветку style к точке перед тем, как мы слили ее с веткой master. Мы хотим вернуться в ветке style в точку перед слиянием с master. Нам необходимо найти последний коммит перед слиянием (рис. [[35](#fig:035)]).

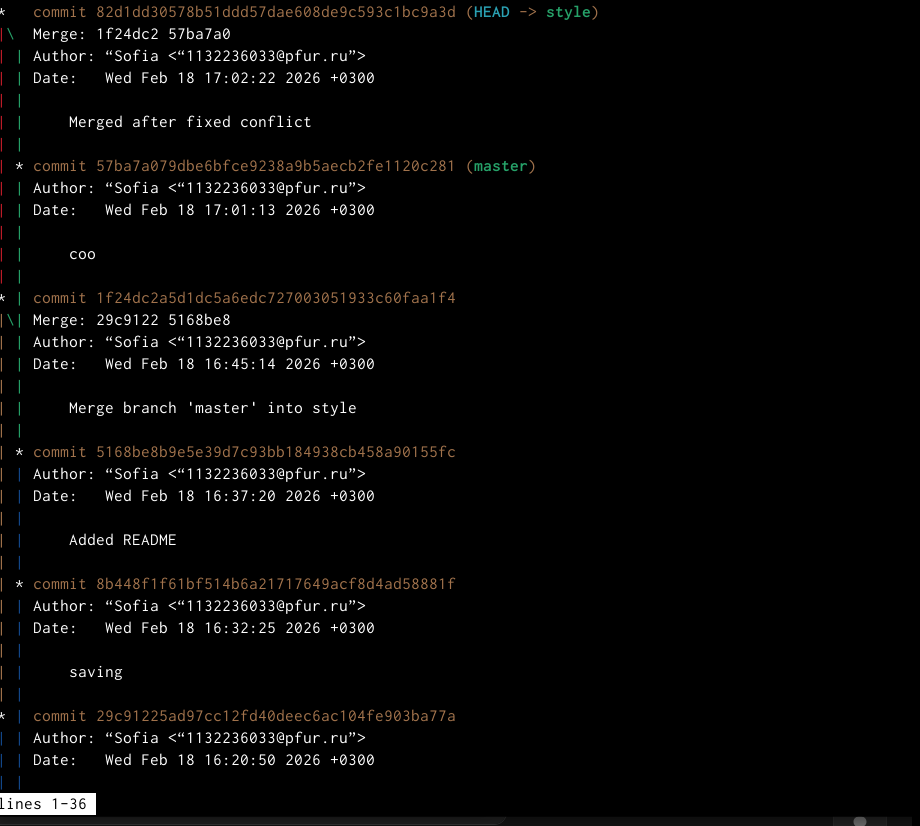


Figure 35: Поиск коммита перед слиянием

Мы видим, что коммит «Updated index.html» был последним на ветке style перед слиянием. Сбросим ветку style к этому коммиту.

Поищем лог ветки style. Увидим, что у нас в истории больше нет коммитов слияний (рис. [[36](#fig:036)]).

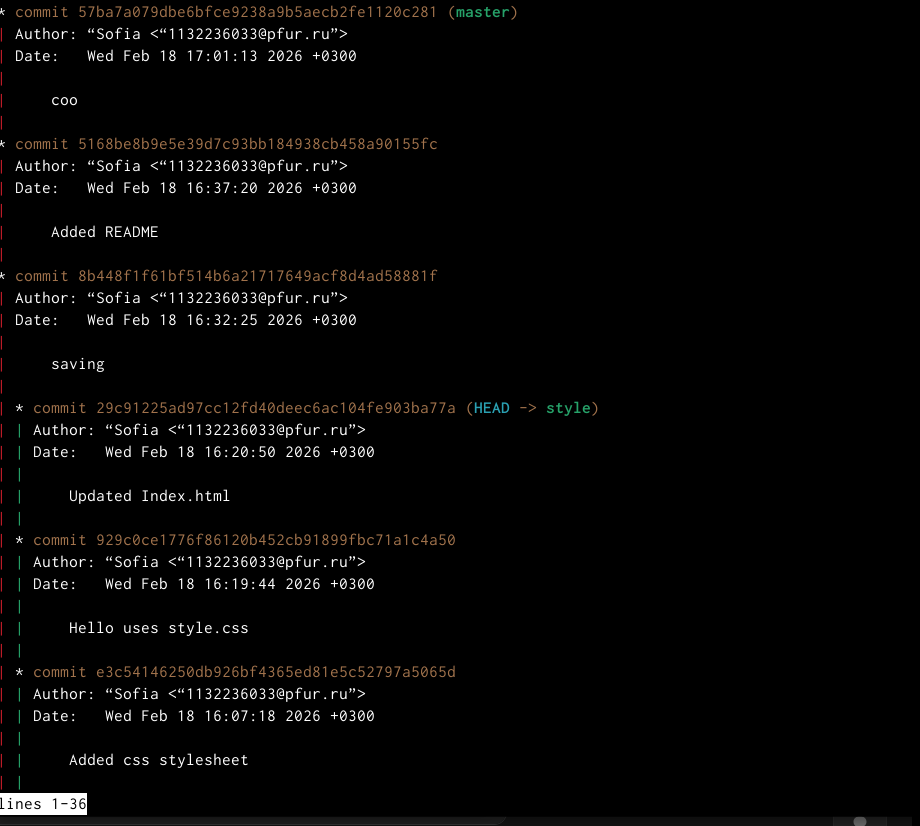


Figure 36: Сброс ветки style

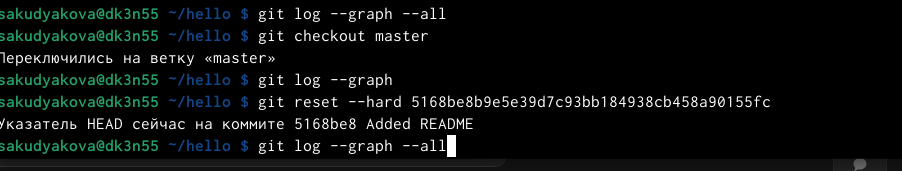
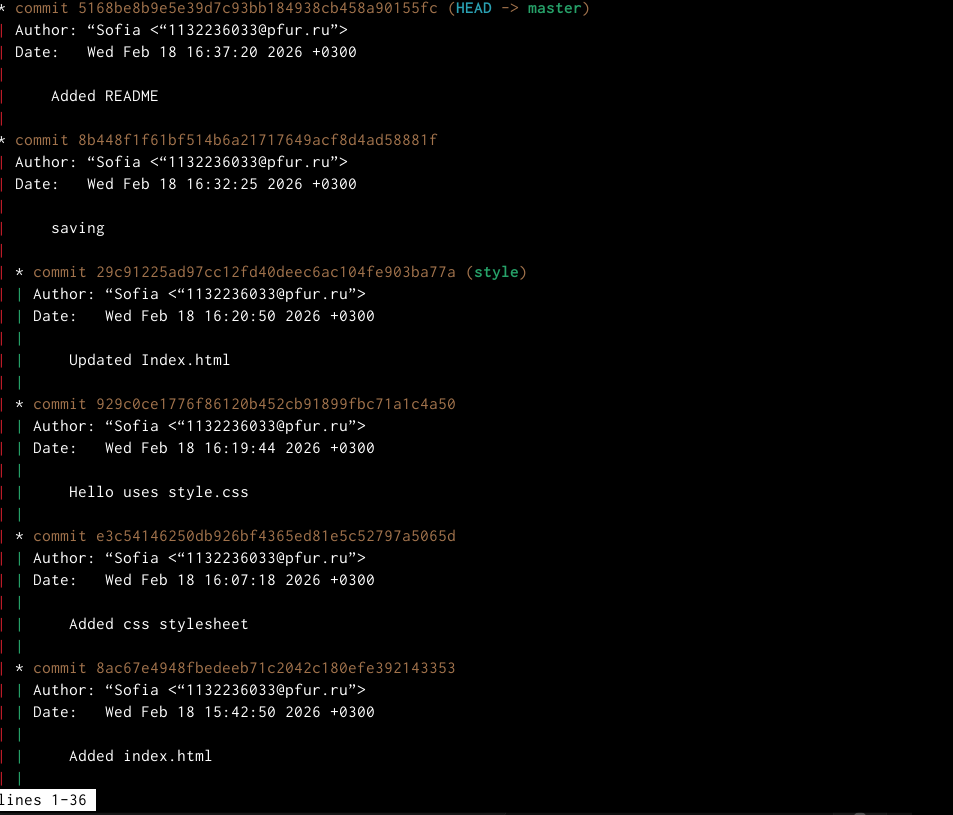
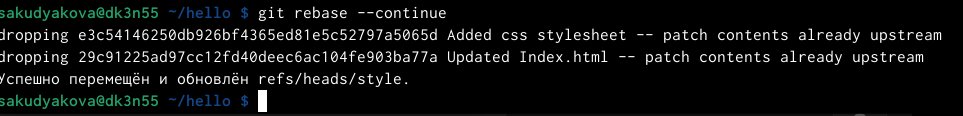
## 4.24 Сброс ветки master

Добавив интерактивный режим в ветку master, мы внесли изменения, конфликтующие с изменениями в ветке style. Давайте вернемся в ветке master в точку перед внесением конфликтующих изменений. Это позволяет нам продемонстрировать работу команды git rebase, не беспокоясь о конфликтах. Просмотрим коммиты ветки master (рис. [[37](#fig:037)]).



Figure 37: Поиск коммита перед конфликтом

Коммит «Added README» идет непосредственно перед коммитом конфликтующего интерактивного режима. Мы сбросим ветку master к коммиту «Added README» (рис. [??]).

## 4.25 Перебазирование

Используем команду rebase вместо команды merge. Мы вернулись в точку до первого слияния и хотим перенести изменения из ветки master в нашу ветку style. На этот раз для переноса изменений из ветки master мы будем использовать команду git rebase вместо слияния (рис. [[38](#fig:039)]).

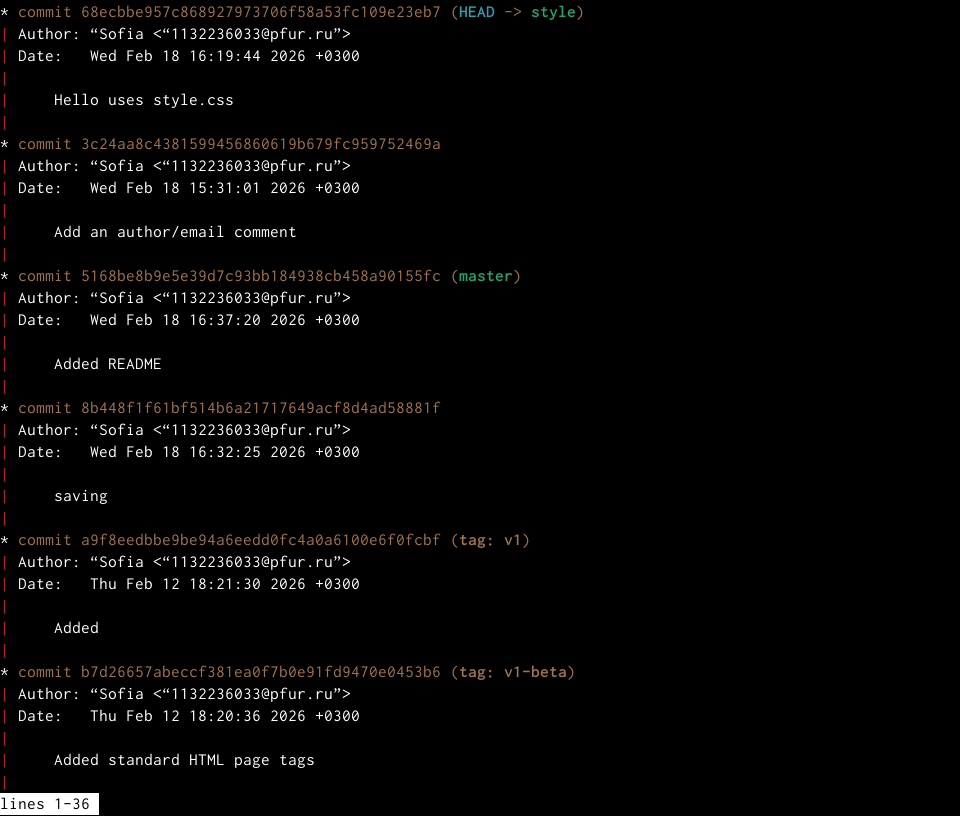
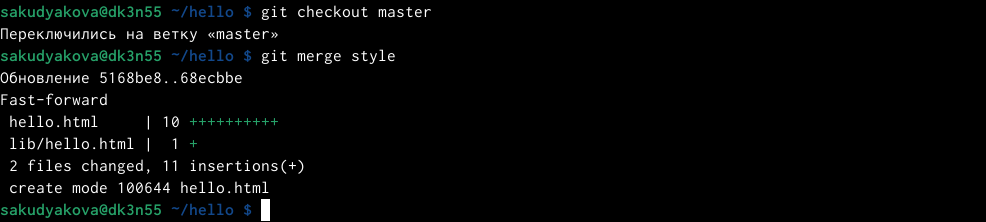
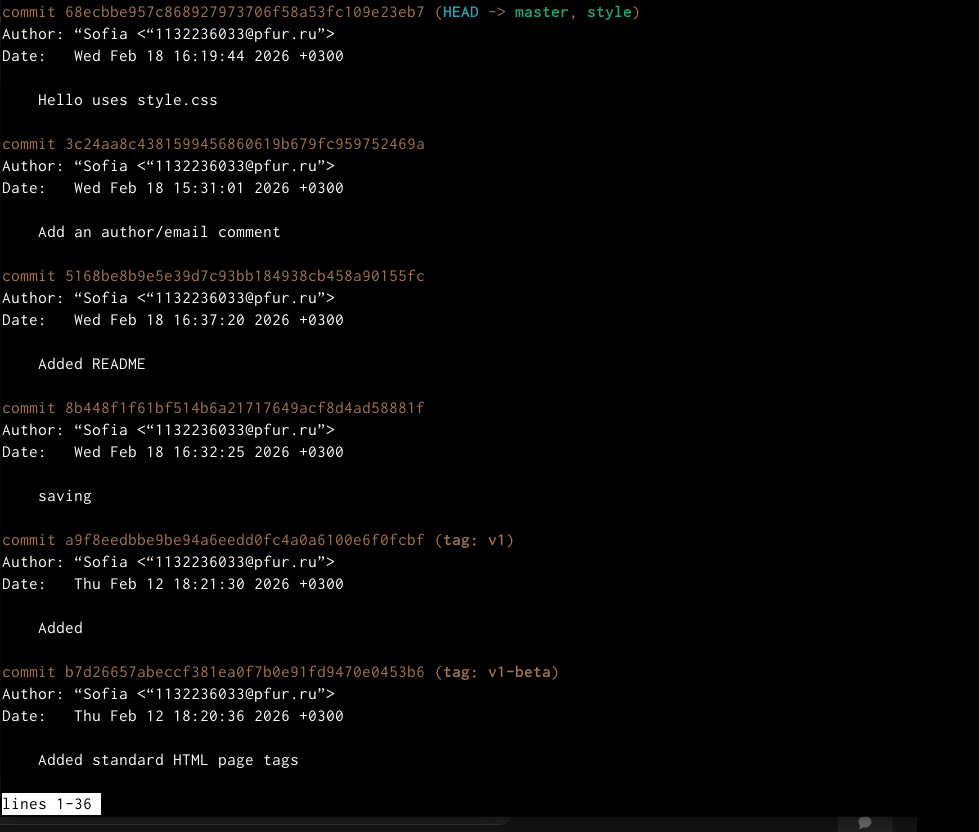


Figure 38: Перебазирование

## 4.26 Слияние в ветку master

Вернемся в ветку master и сольем ветку style в неё с помощью команды git merge(рис. [??]).

## 4.27 Клонирование репозиториев

Перейдем в наш рабочий каталог и сделаем клон репозитория hello. Просмотрев его, увидим список всех файлов на верхнем уровне оригинального репозитория README.md, index.html и lib (рис. [[39](#fig:041)]).

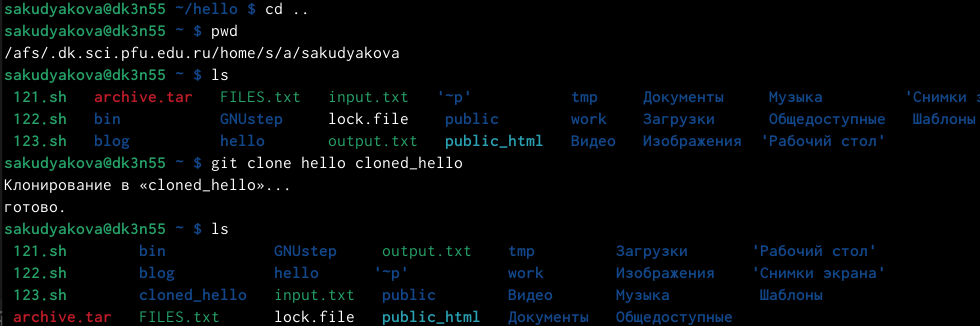
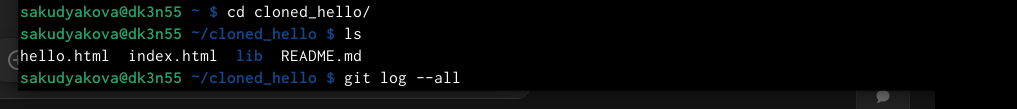
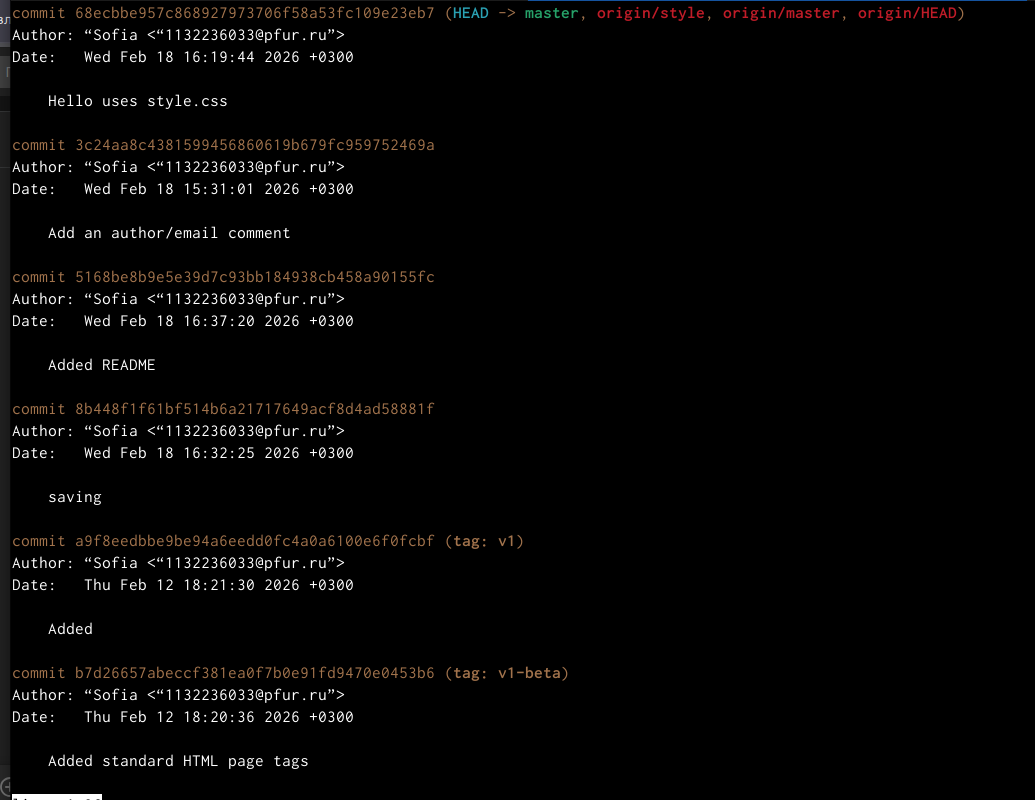


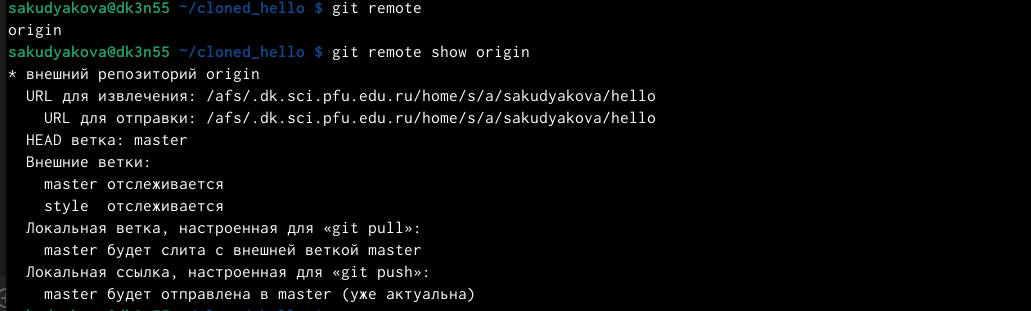
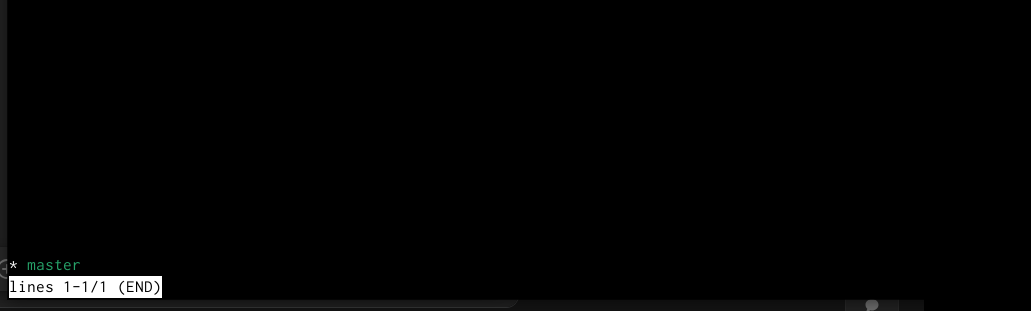
Figure 39: Клонирование репозиториев

Затем просмотрим историю репозитория и увидим список всех коммитов в новый репозиторий, и он совпадает с историей коммитов в оригинальном репозитории. Единственное отличие в названиях веток (рис. [??]).

## 4.28 Что такое origin?

Клонированный репозиторий знает об имени по умолчанию удаленного репозитория. Посмотрим, подробную информацию об имени по умолчанию. Давайте посмотрим, можем ли мы получить более по- дробную информацию об имени по умолчанию (рис. [??]).

## 4.29 Удаленные ветки

Посмотрим на ветки, доступные в нашем клонированном репозитории. Можно увидеть, что в списке только ветка master. Для того, чтобы увидеть все ветки используем опцию -a (рис. [??]).

Просмотр веток 

## 4.30 Изменение оригинального репозитория

Перейдем в репозиторий hello. Внесем изменения в файл README.md. Затем добавим их в репозиторий (рис. [[40](#fig:045)]).

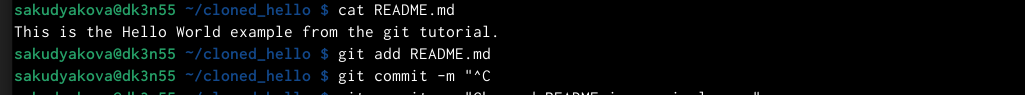
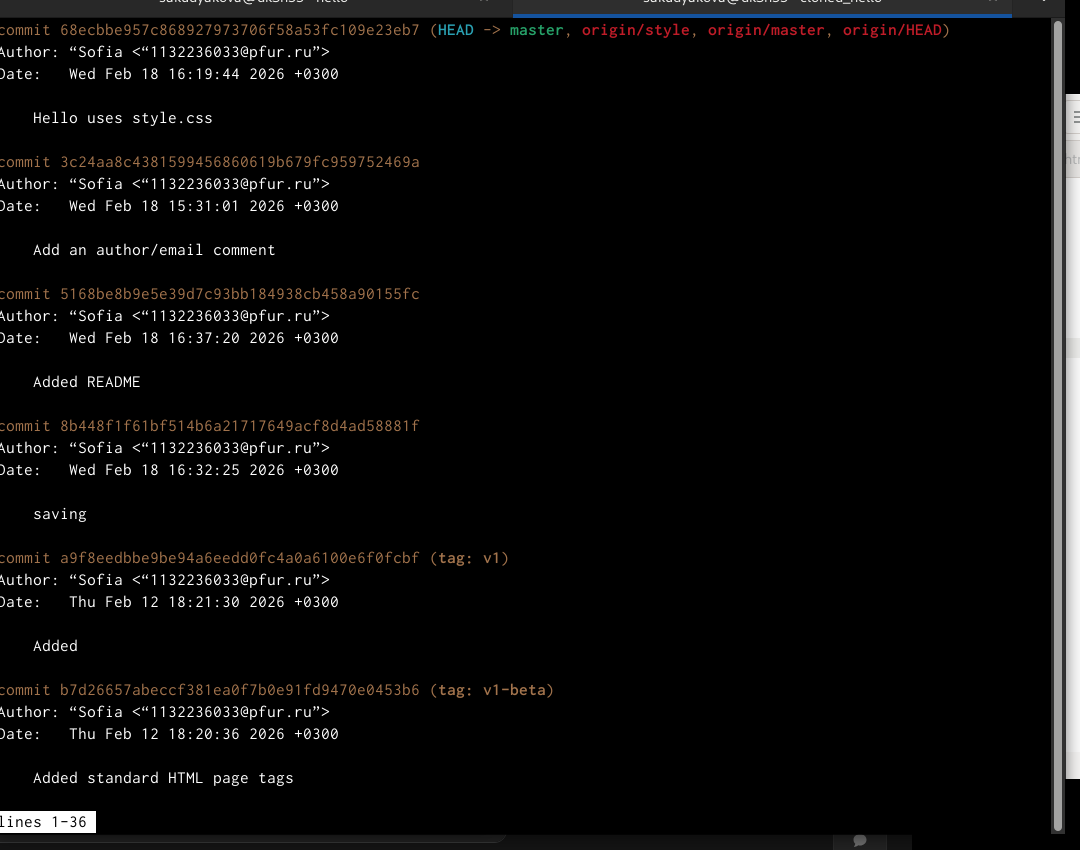


Figure 40: Изменение оригинального репозитория

Перейдём в клон репозитория и используем команду git fetch, которая будет извлекать новые коммиты из удаленного репозитория, но не будет сливать их с наработками в локальных ветках (рис. [??]).

## 4.31 Слияние извлеченных изменений

Сольем внесённые изменения в главную ветку с помощью git pull, которая является объединением fetch и merge в одну команду (рис. -[41](#fig:047)).

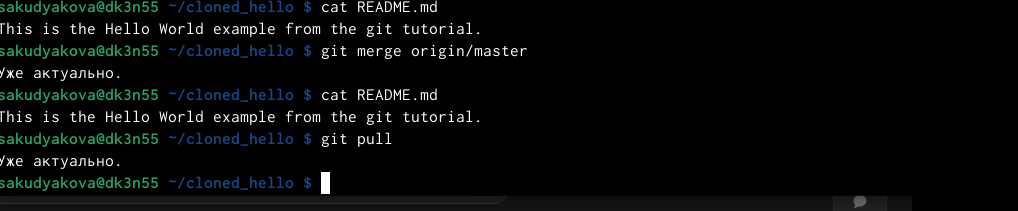
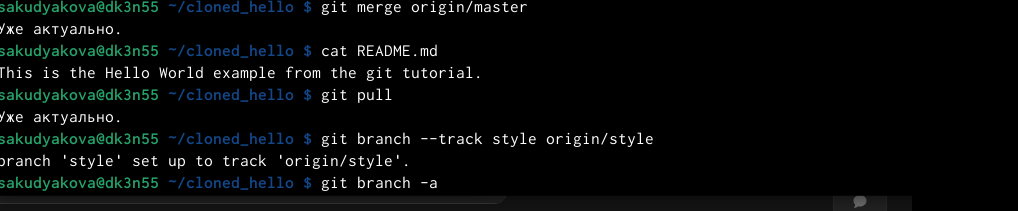
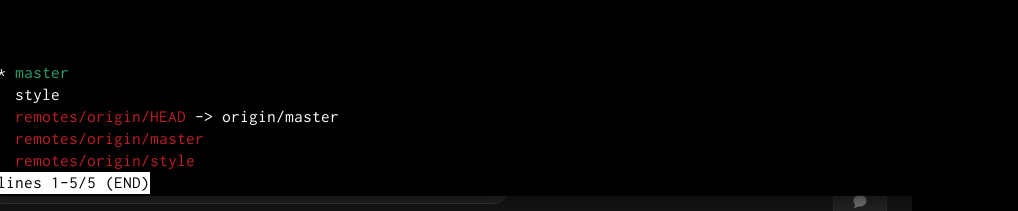
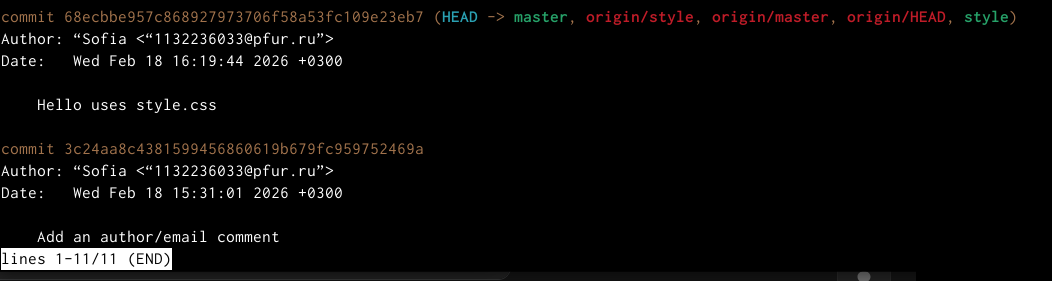


Figure 41: Слияние извлеченных изменений

## 4.32 Добавление ветки наблюдения

Добавим локальную ветку, которая отслеживает удаленную ветку, теперь мы можем видеть ветку style в списке веток и логе (рис. [??]).

 Добавление ветки наблюдения  

## 4.33 Создание чистого репозитория

Как правило, репозитории, оканчивающиеся на .git являются чистыми репозиториями. Создадим такой в рабочем каталоге. Затем добавим репозиторий hello.git к нашему оригинальному репозиторию (рис. [[42](#fig:049)]).

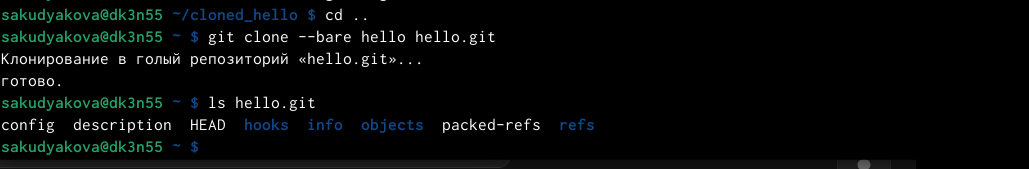


Figure 42: Создание чистого репозитория

## 4.34 Отправка и извлечение изменений

Так как чистые репозитории, как правило, расшариваются на каком-нибудь сетевом сервере, нам необходимо отправить наши изменения в другие репозитории. Начнем с создания изменения для отправки. Отредактируем файл README.md и сделаем коммит, затем отправим изменения в общий репозиторий. Затем извлечем изменения из общего репозитория(рис. [[43](#fig:050)], [[44](#fig:051)]).

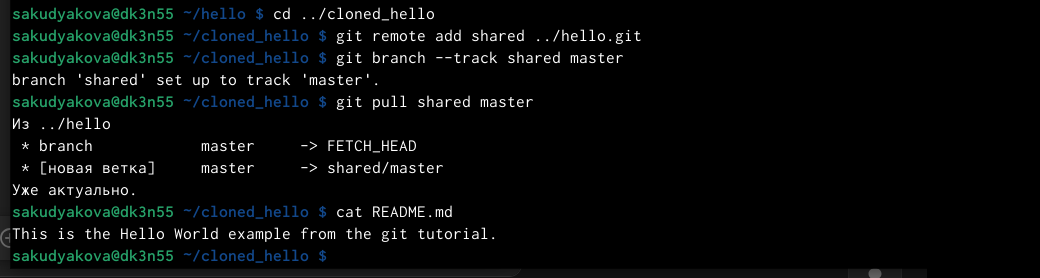


Figure 43: Отправка изменений

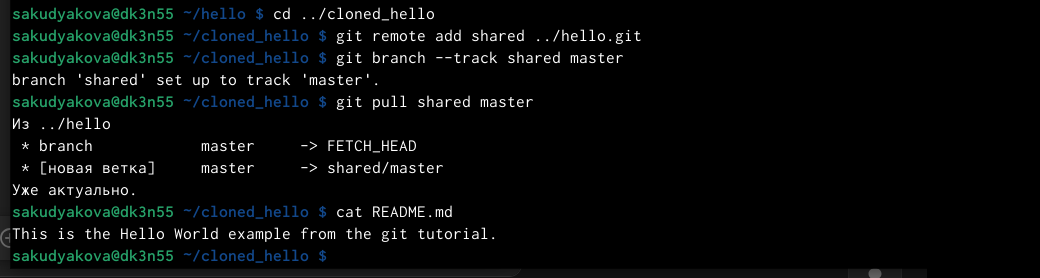


Figure 44: Извлечение изменений

# 5 Выводы

В процессе выполнения данной лабораторной работы я приобрела практические навыки работы с Git.

# Список литературы

::: [Математическое моделирование](https://esystem.rudn.ru/course/view.php?id=5935) :::