Отчёт по лабораторной работе №1

Математическое моделирование

Бансимба Клодели Дьегра

Содержание

Список иллюстраций

Список таблиц

# 1 Цель работы

Целью данной работы является приобретение практических навыков установки операционной системы на виртуальную машину, настройки минимально необходимых для дальнейшей работы сервисов

# 2 Выполнение лабораторной работы

## 2.1 Подготовка

Если вы никогда ранее не использовали git, для начала вам необходимо осуществить установку. Выполните следующие команды, чтобы git узнал ваше имя и электронную почту (рис. fig. 1)..

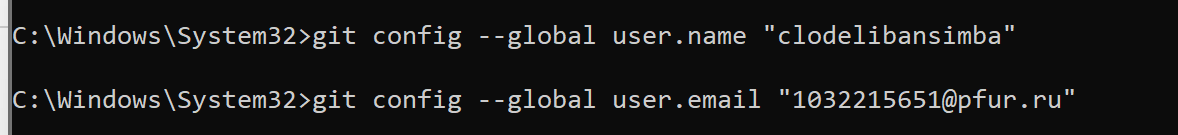


Рис. 1: становка имени и электронной почты

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. fig. 2) и (рис. fig. 3) .

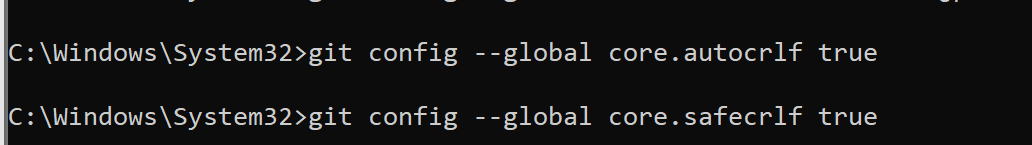


Рис. 2: араметры установки окончаний строк

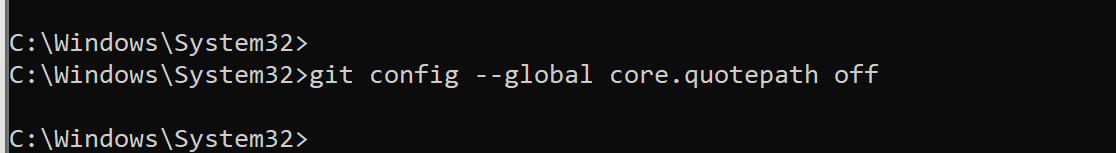


Рис. 3: становка отображения unicode

## 2.2 Создание проекта

Создадим пустой каталог hello, а в нём файл с именем hello.html. Затем создадим git репозиторий из этого каталога, выполнив команду git init. Добавим файл в репозиторий и проверим статус, который сообщает, что коммитить нечего (рис. fig. 4), (рис. fig. 5).

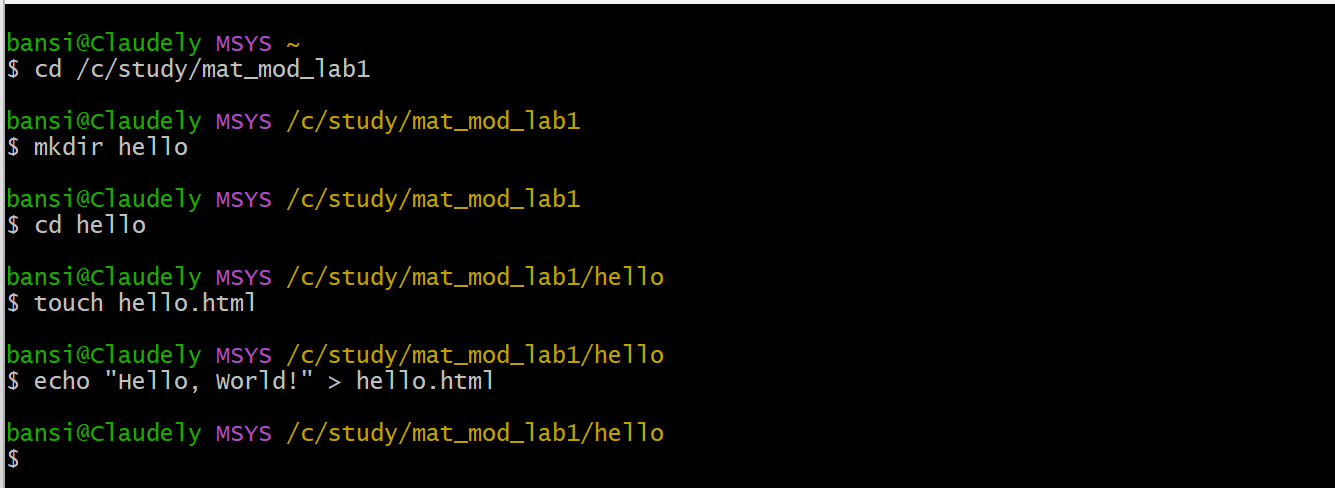


Рис. 4: Создание репозитория

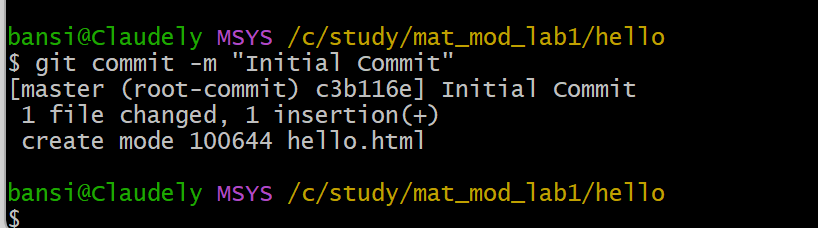


Рис. 5: Добавление файла в репозиторий

## 2.3 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. Теперь проиндексируем изменения и снова посмотрим статус, в нём 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. fig. 6) и (рис. fig. 7).

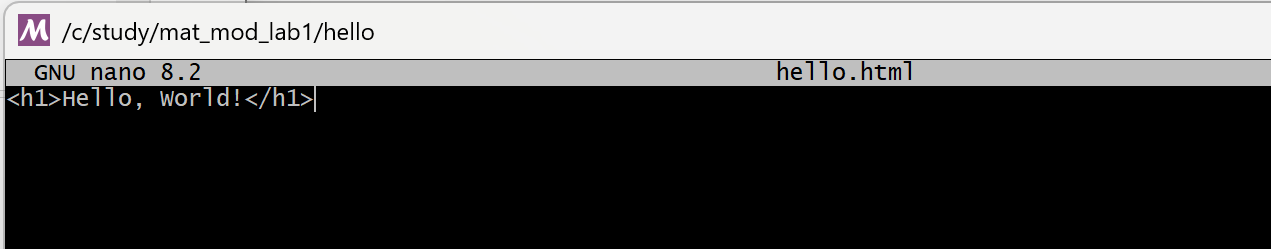


Рис. 6: Внесение изменений в содержимое репозитория

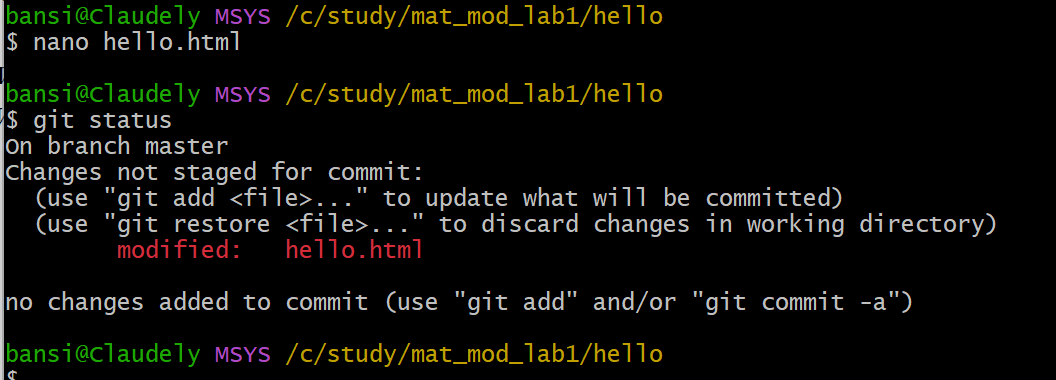


Рис. 7: Внесение изменений в содержимое репозитория

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

и

(рис. fig. 8).

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

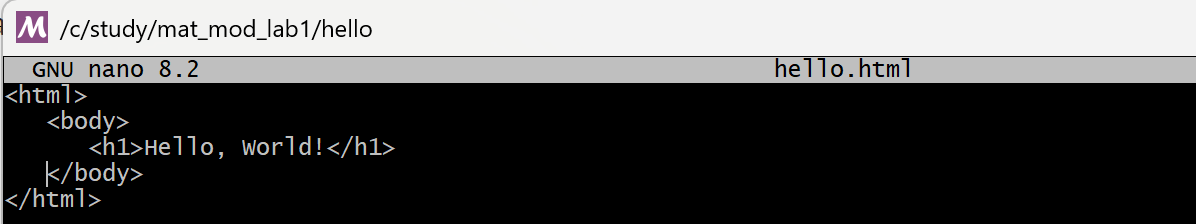


Рис. 8: обавьте стандартные теги страницы

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

) к странице «Hello, World» (рис. fig. 9).

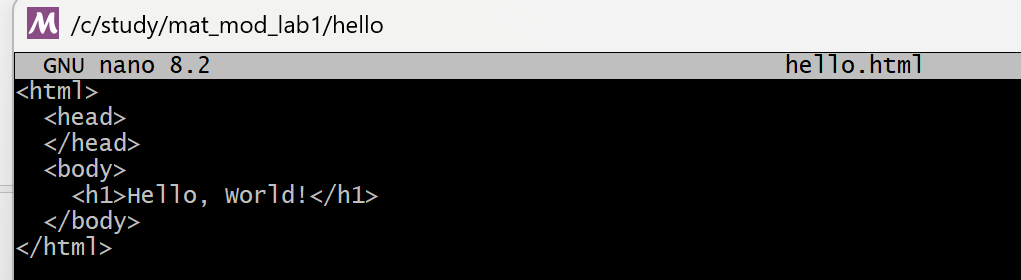


Рис. 9: обавьте стандартные теги страницы

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) является непроиндексированным. Произведем коммит проиндексированного изменения, затем проиндексируем оставшееся изменение, посмотрим статус и прокоммитим его(рис. fig. 10).

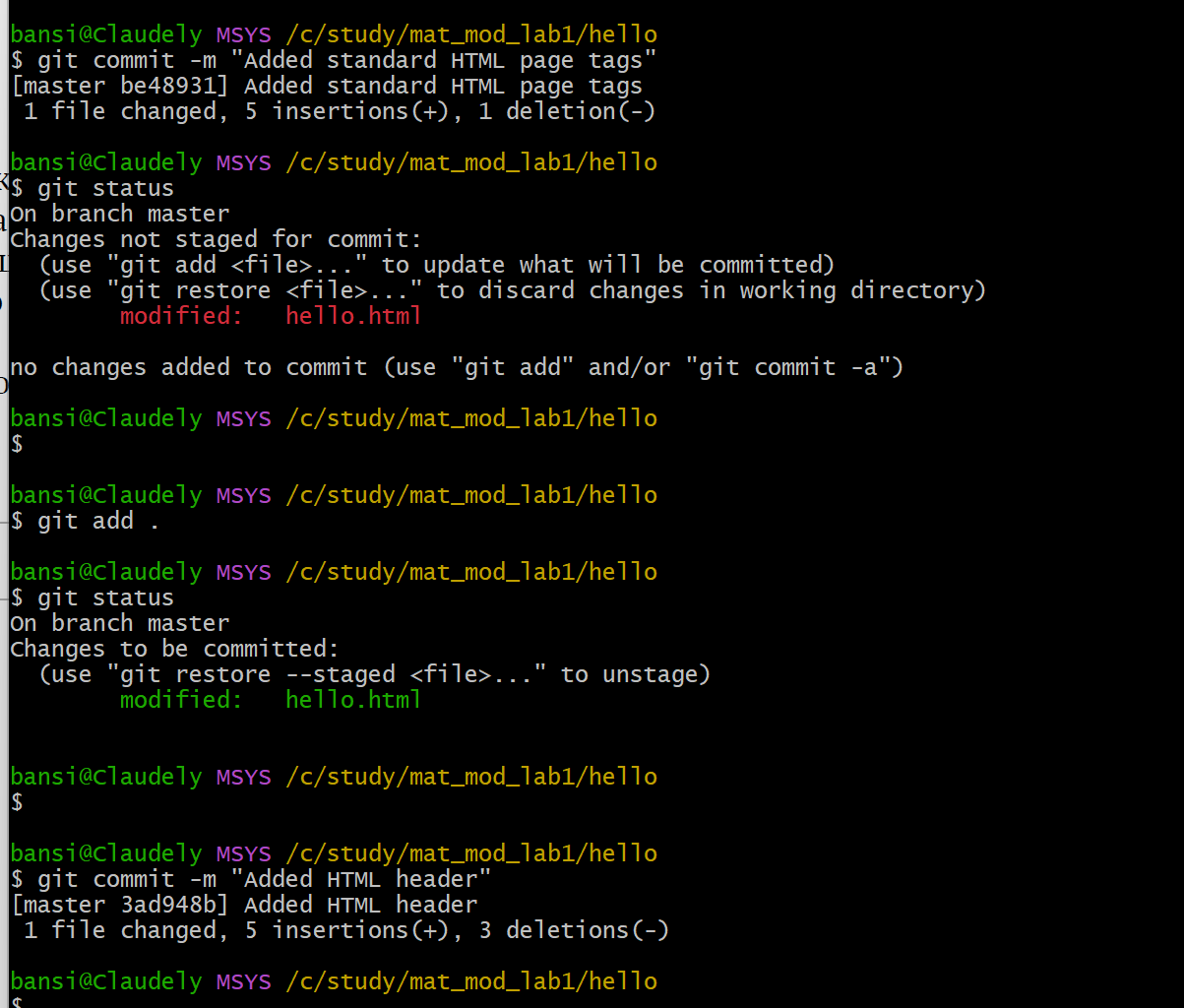


Рис. 10: Внесение нескольких изменений в содержимое репозитория

## 2.4 История

Получим список произведенных изменений (рис. fig. 11): git log

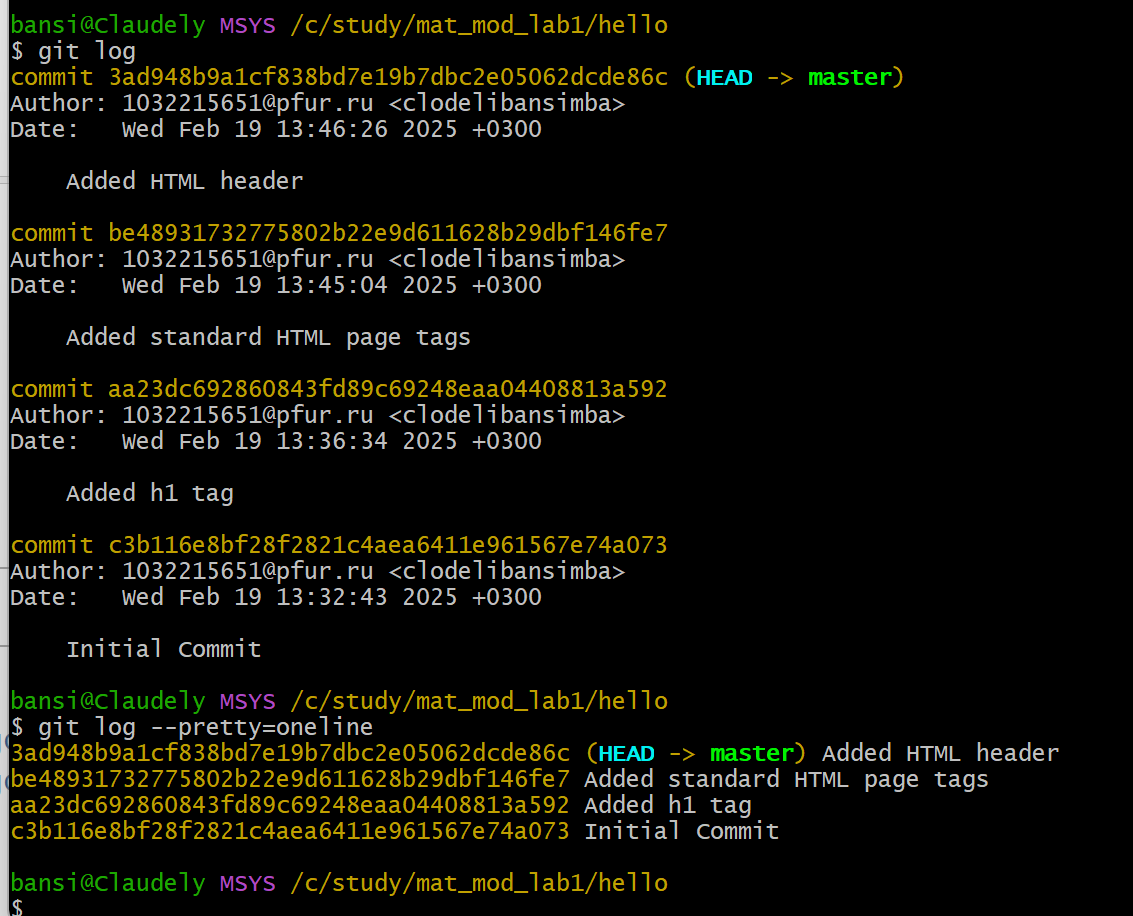


Рис. 11: список произведенных изменений

Однострочный формат истории: git log –pretty=oneline

Есть много вариантов отображения лога.

git log –pretty=oneline –max-count=2 git log –pretty=oneline –since=‘5 minutes ago’ git log –pretty=oneline –until=‘5 minutes ago’ git log –pretty=oneline –author=clodeli bansimba git log –pretty=oneline –all

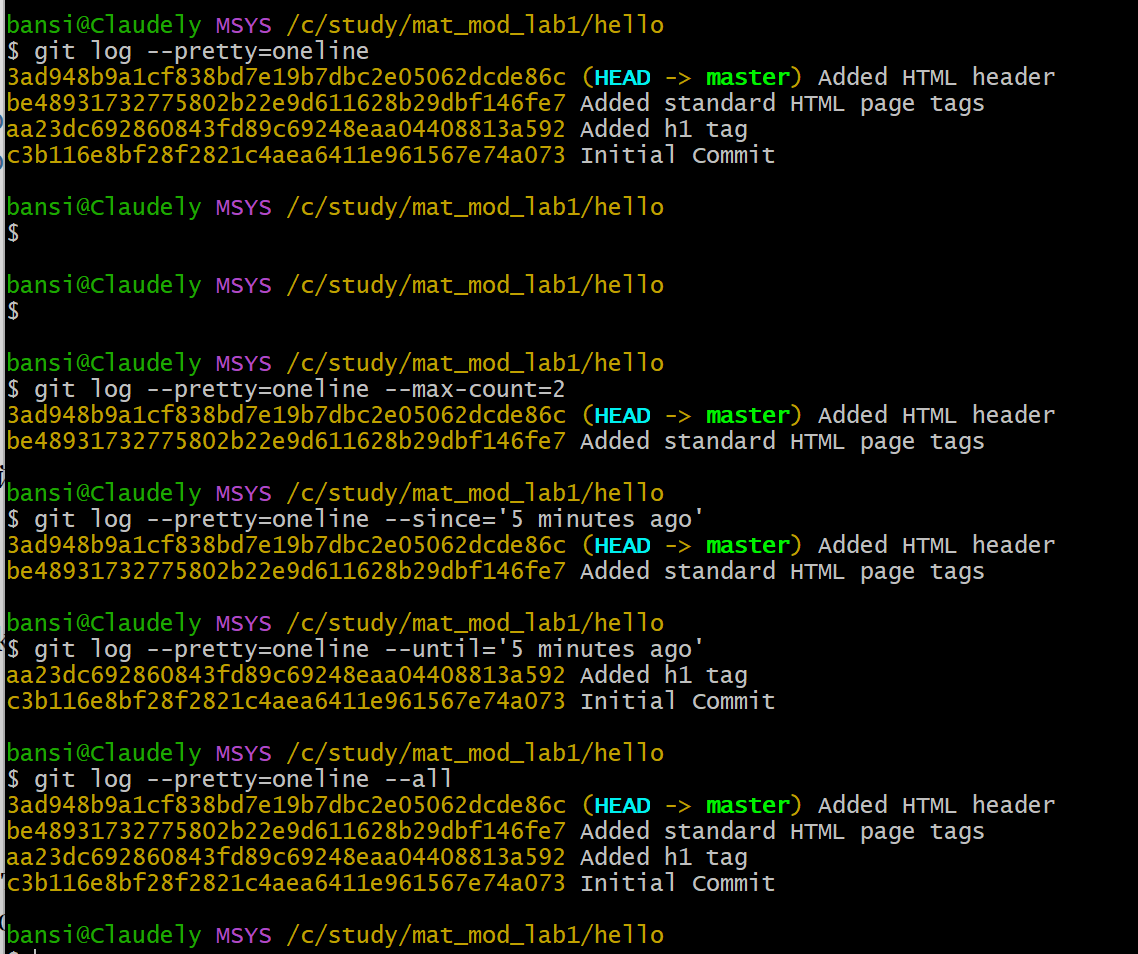


Рис. 12: Однострочный формат истории и Есть много вариантов

## 2.5 Получение старых версий

Изучим данные лога и найдем там хэш первого коммита, используя его вернемся к первой верссии и просмотрим файл hello.html, действительно, увидим первую версию. Затем вернемся к последней версии в ветке master и вновь посмотрим на файл(рис. fig. 13), (рис. fig. 14).

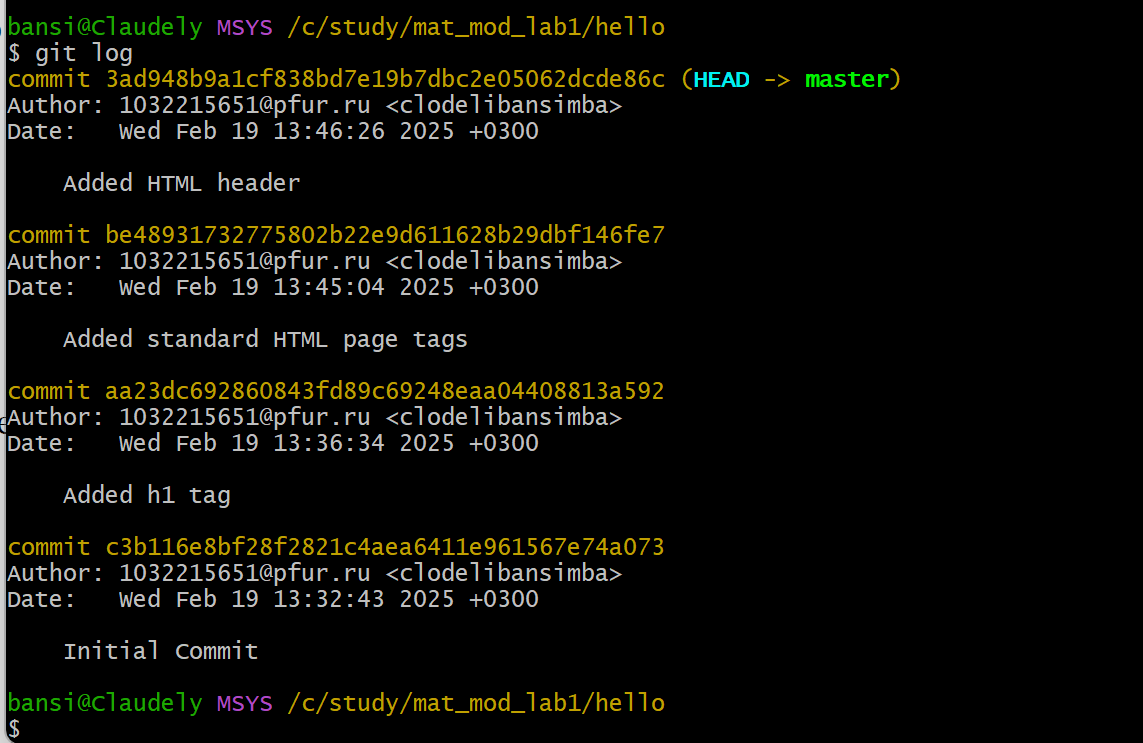


Рис. 13: Просмотр разных версий репозитория

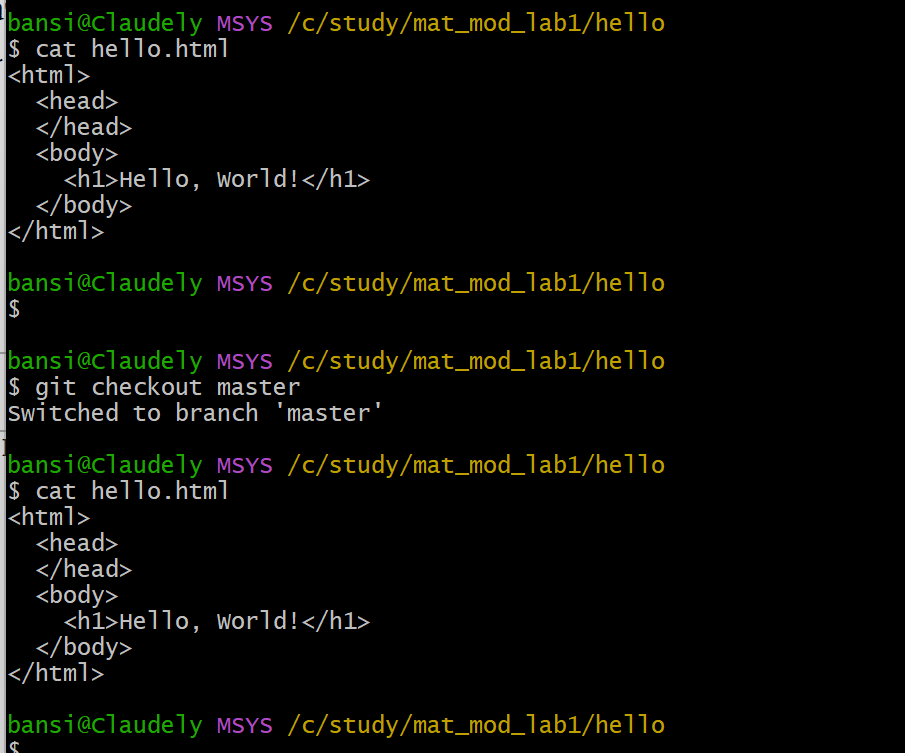


Рис. 14: Просмотр разных версий репозитория

## 2.6 Создание тегов версий

Назовем текущую версию страницы hello первой (v1). Создадим тег первой версии и используем его для того чтобы венуться к предыдущей, которой также присвоим тег (рис. fig. 15).

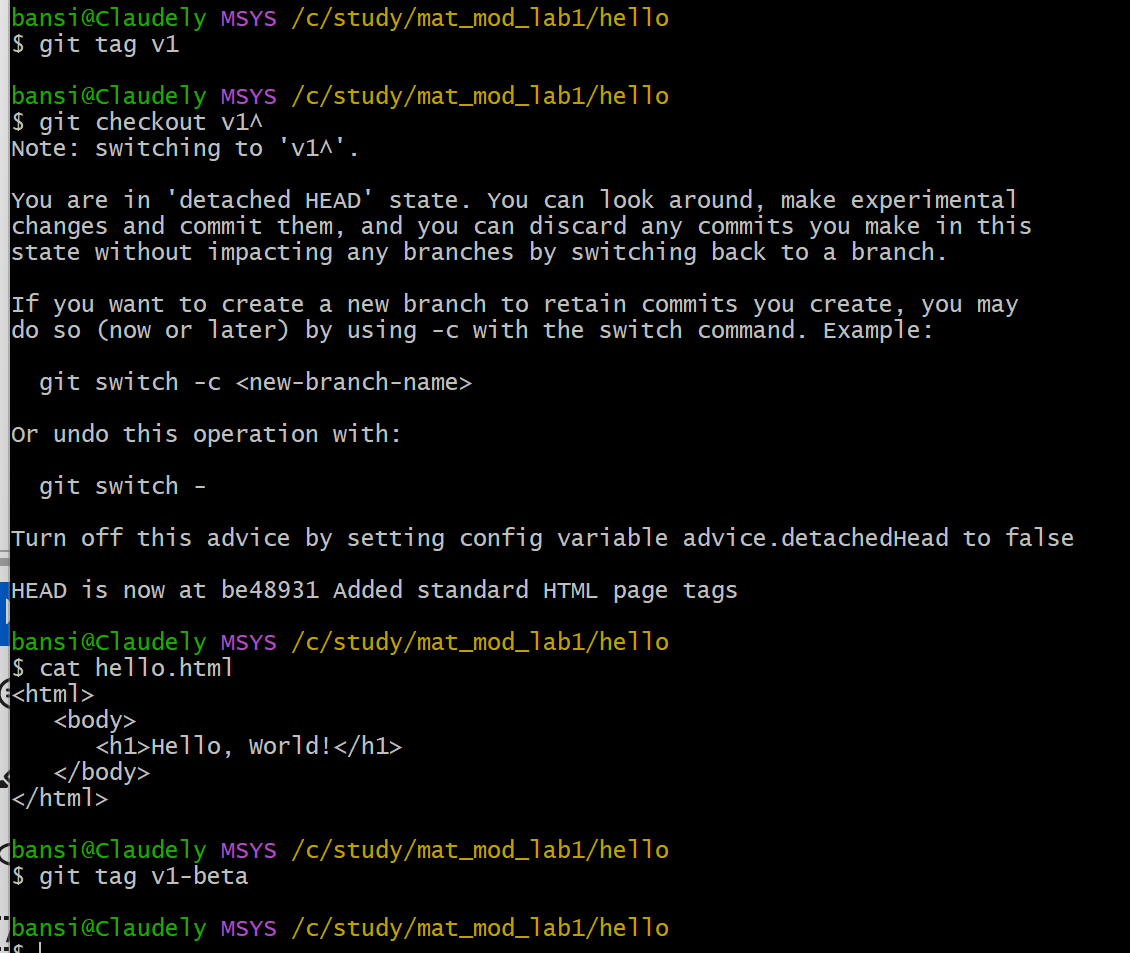


Рис. 15: Создание тегов версий

Переключимся по тегам между двумя отмеченными версиями. Просмотрим все доступные теги(их два) и посмотрим теги в логе(рис. fig. 16).

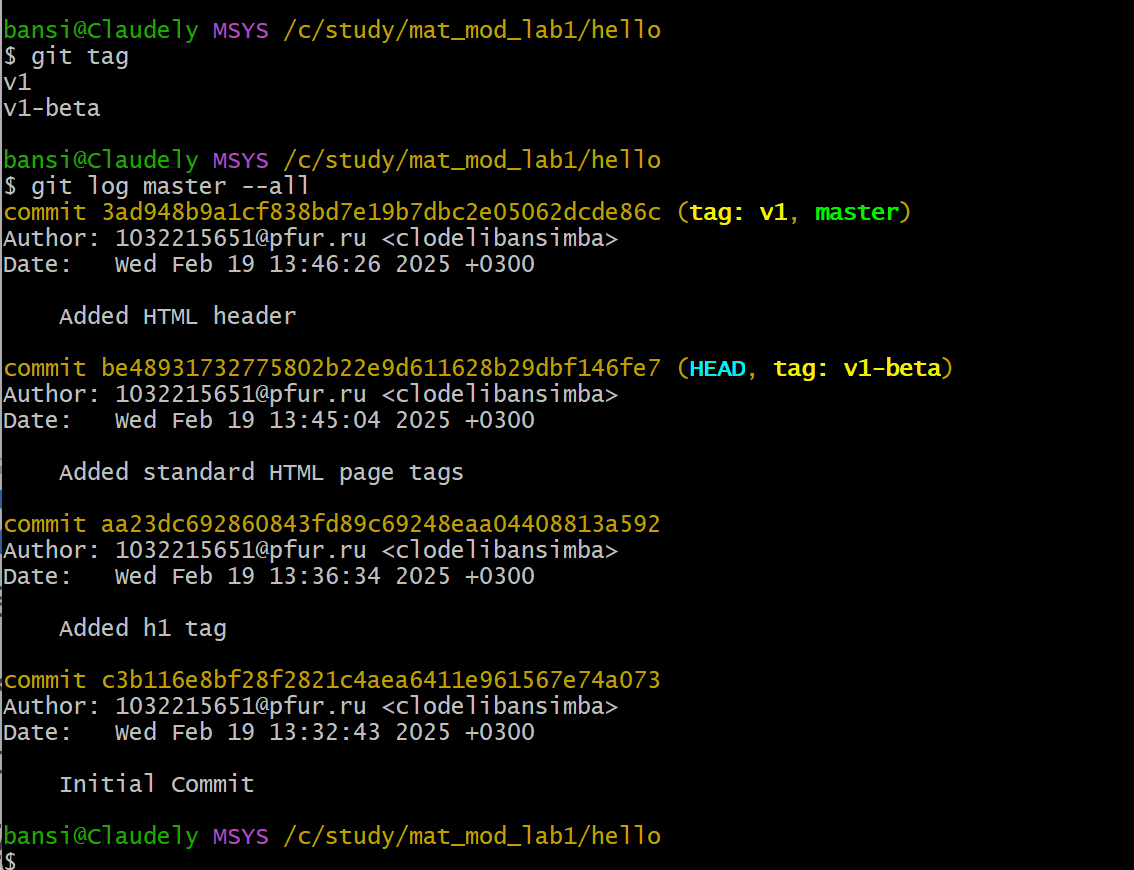


Рис. 16: Переключение по имени тега и просмотр доступных тегов

## 2.7 Отмена локальных изменений (до индексации)

Убдеимся, что мы находимся на последнем коммите ветки master (рис. fig. 17).

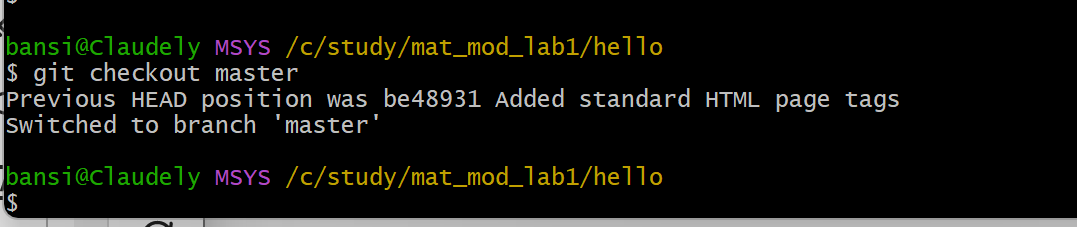


Рис. 17: Отмена локальных изменений (до индексации)

и внесем изменение в файл hello.html в виде нежелательного комментария (рис. fig. 18).



Рис. 18: изменение в файл

Затем проверим статус, увидим, что изменения ещё не проиндексированы (рис. fig. 19).



Рис. 19: изменение в файл

Используем команду git checkout для переключения версии файла hello.html в репозитории (рис. fig. 20).

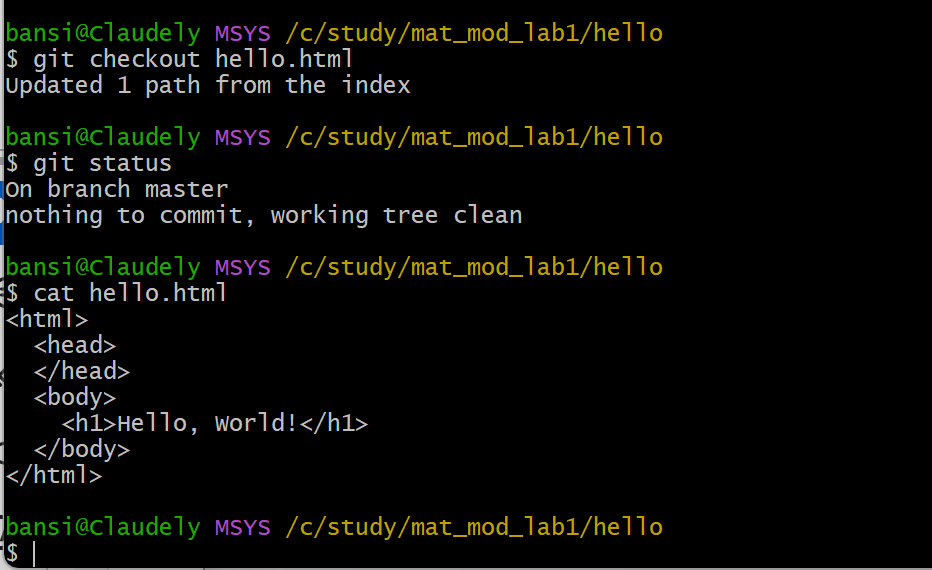


Рис. 20: git checkout

## 2.8 Отмена проиндексированных изменений (перед коммитом)

### 2.8.1 Измените файл и проиндексируйте изменения

Внесем изменение в файл hello.html в виде нежелательного комментария (рис. fig. 21).

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

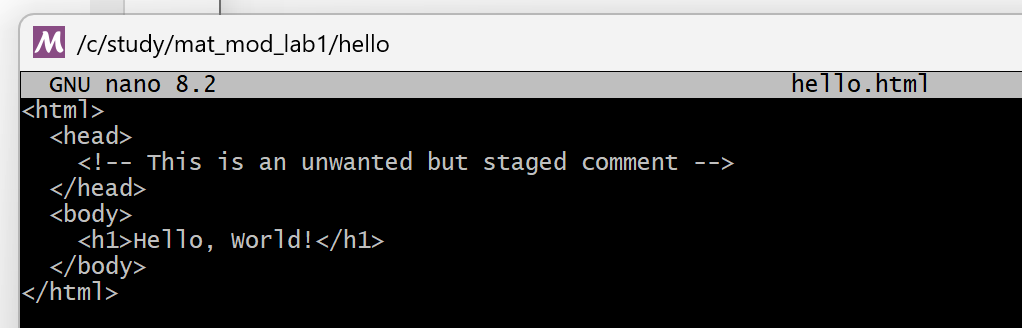


Рис. 21: изменение в файл hello.html

Проиндексируем это изменение и проверим состояние (рис. fig. 22). git status

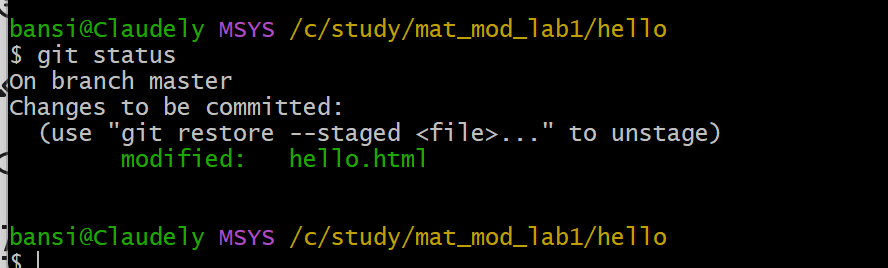


Рис. 22: git status

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. И переключимся на последнюю версию коммита, посмотрев статус увидим, что наш каталог опять чист (рис. fig. 23), (рис. fig. 24).

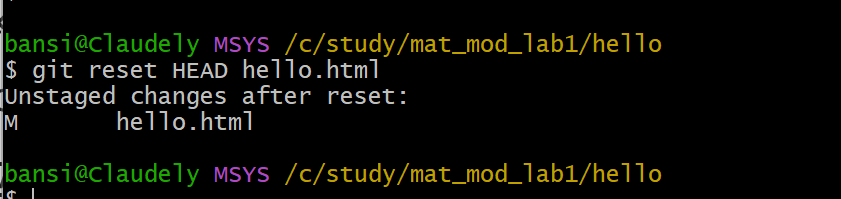


Рис. 23: Отмена проиндексированных изменений (перед коммитом)

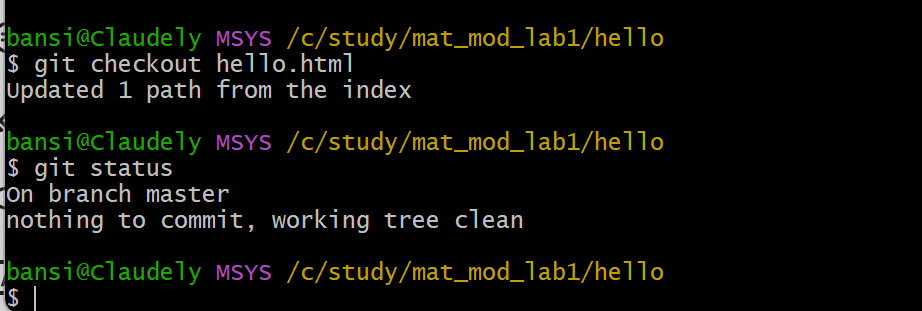


Рис. 24: Отмена проиндексированных изменений (перед коммитом)

## 2.9 Отмена коммитов

Изменим файл hello.html на следующий (рис. fig. 25) (рис. fig. 26).

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

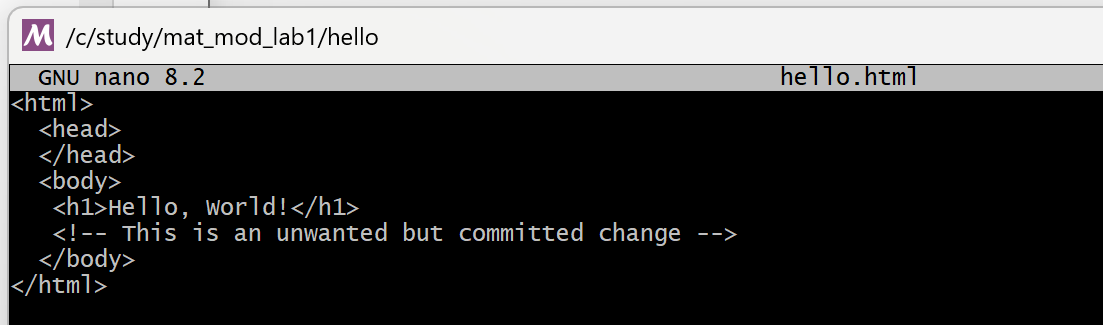


Рис. 25: Изменим файл hello.html

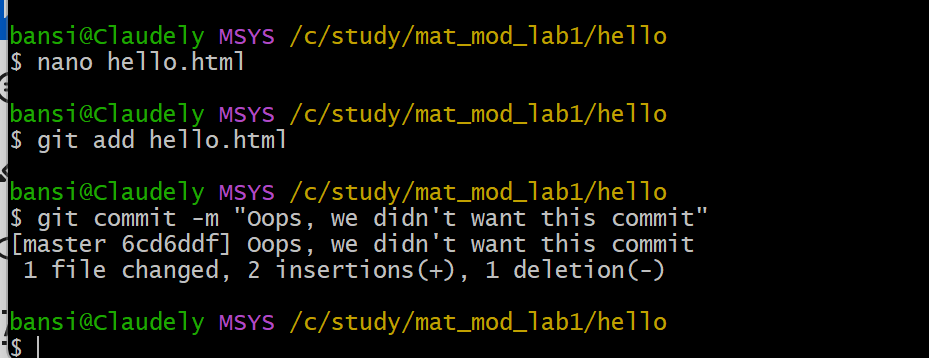


Рис. 26: Изменим файл hello.html

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 наш репозиторий(рис. fig. 27).



Рис. 27: Отмена коммитов

## 2.10 Удаление коммиттов из ветки

Удалим последние два коммита с помощью сброса, сначала отметим последний коммит тегом, чтобы его можно было потом найти. Используем команду git reset, чтобы вернуться к версии до этих коммитов. Теперь в логе их нет, но если посмотреть логи с опцией –all можно всё ещё их увидеть, но метка HEAD находится на нужной нам версии.

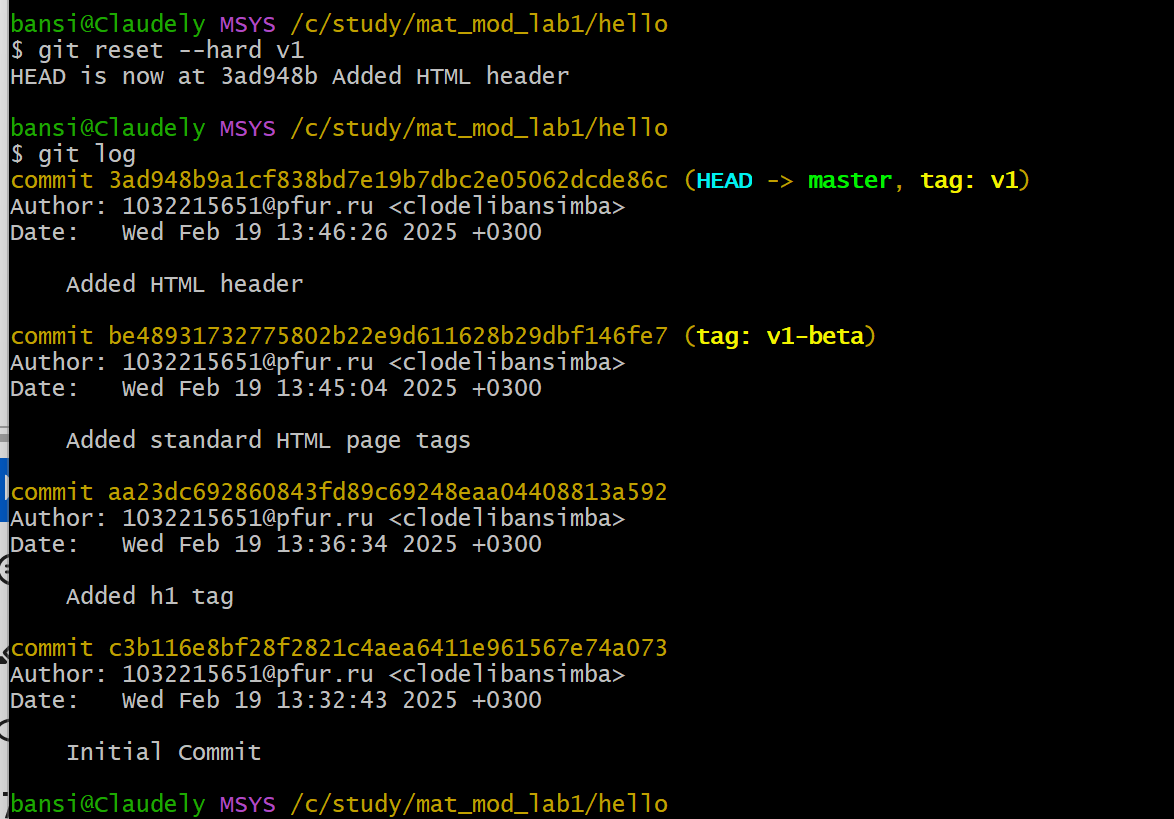


Рис. 28: Удаление коммиттов из ветки

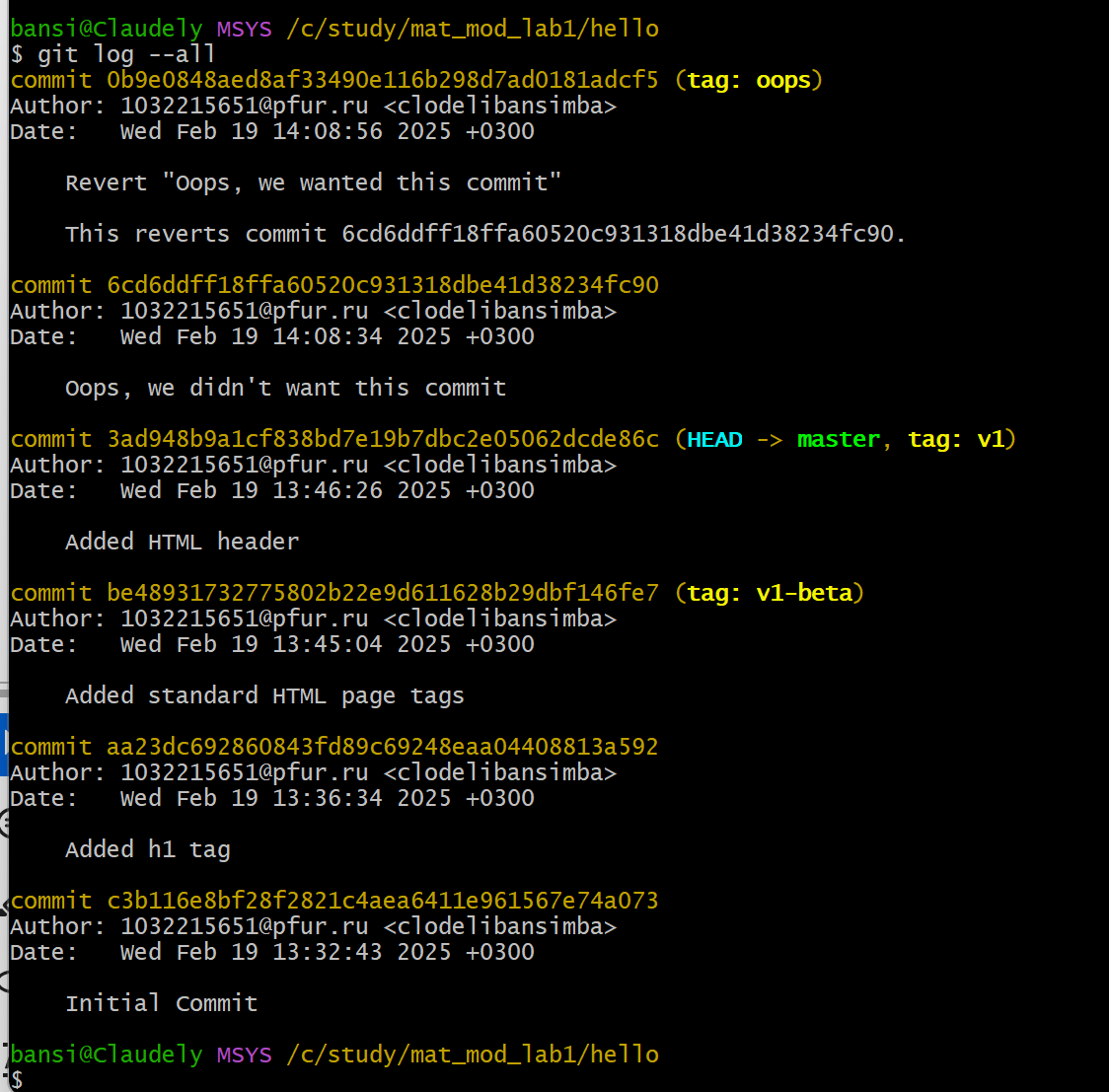


Рис. 29: Удаление коммиттов из ветки

## 2.11 Удаление тега oops

Удалим тег oops и коммиты, на которые он ссылался, сборщиком мусора. Теперь этот тег не отображается в репозитории 030).

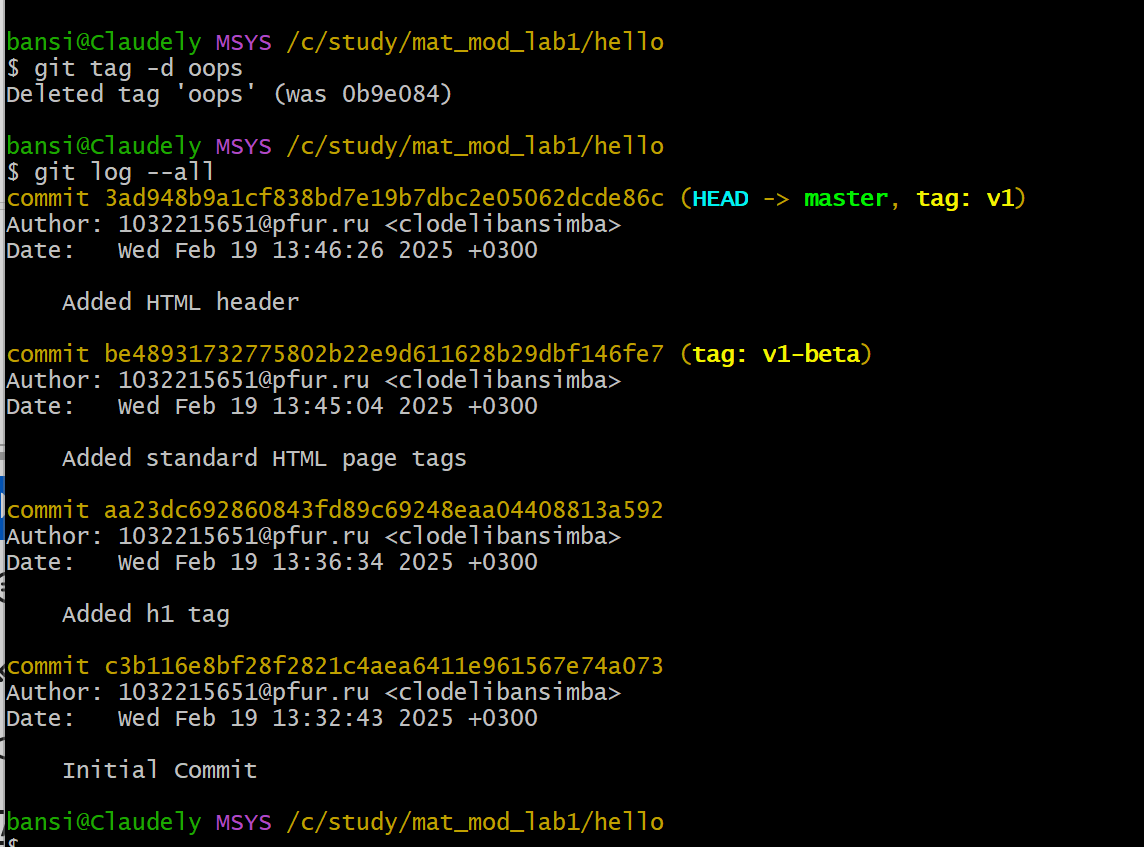


Рис. 30: Удаление тега oops

## 2.12 Внесение изменений в коммиты

### 2.12.1 Измените страницу, а затем сделайте коммит

Добавим в страницу комментарий автора (рис. fig. 31)

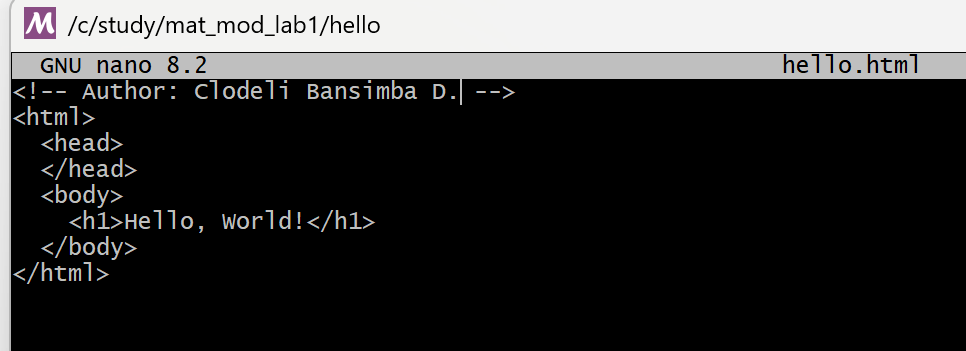


Рис. 31: автора

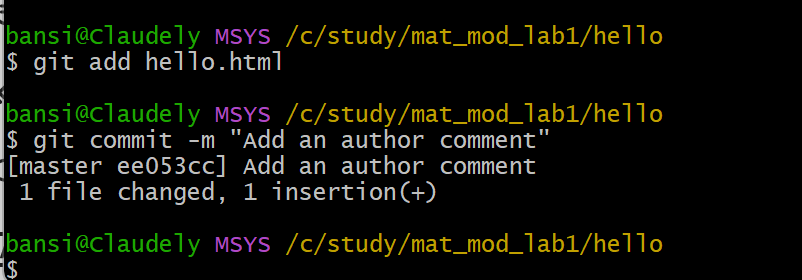


Рис. 32: Commit

### 2.12.2 Необходим email

комментарий должен включать электронную почту автора (рис. fig. 33).



Рис. 33: Email

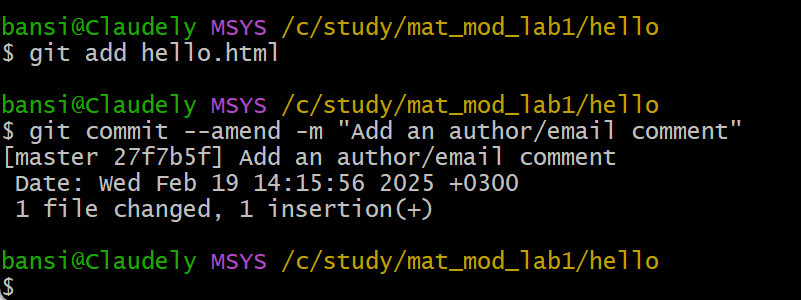


Рис. 34: Email

## 2.13 Просмотр истории

git log

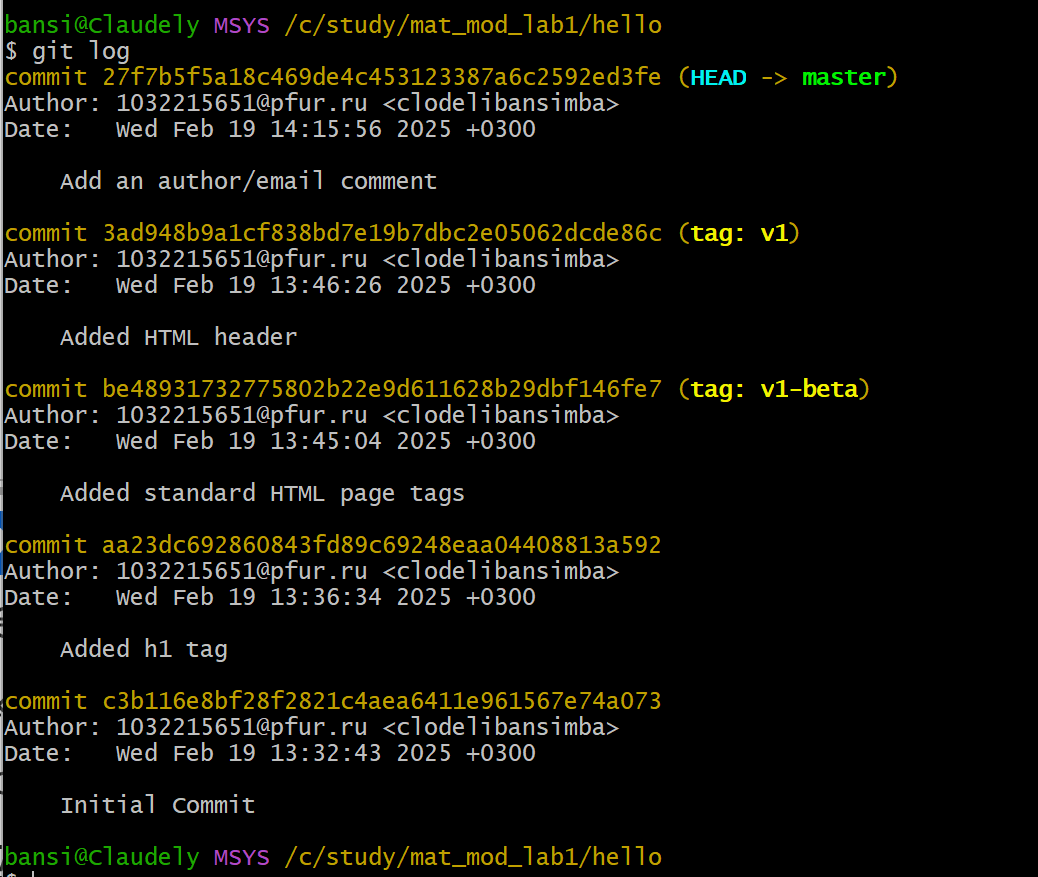


Рис. 35: Email

Мы можем увидеть, что оригинальный коммит «автор» заменен коммитом «автор/email». Этого же эффекта можно достичь путем сброса последнего коммита в ветке, и повторного коммита новых изменений (рис. fig. 35).

## 2.14 Перемещение файлов

Переместим наш файл в каталог lib. Для этого создадим его и используем команду git mv, сделаем коммит этого пермещения (рис. fig. 36).

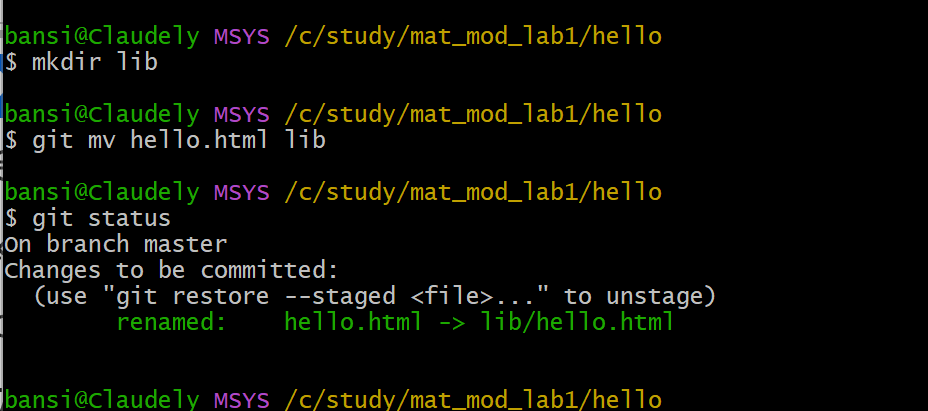


Рис. 36: Перемещение файлов

## 2.15 Второй способ перемещения файлов

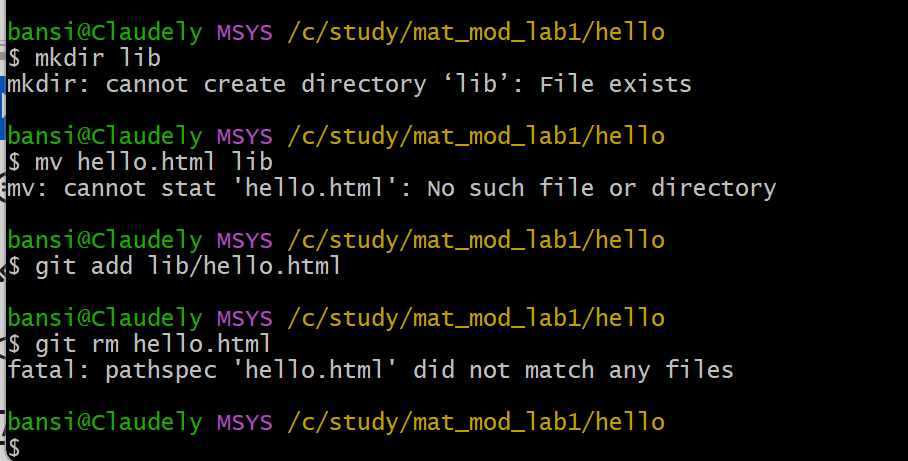


Рис. 37: Второй способ перемещения файлов

сделаем коммит этого перемещения (рис. fig. 38)

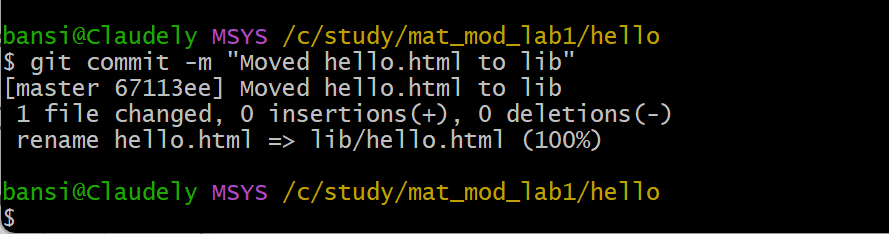


Рис. 38: Коммит в новый каталог

## 2.16 Подробнее о структуре

Добавим файл index.html в наш репозиторий (рис. fig. 39)

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



Рис. 39: index.html

Добавим файл и сделаем коммит (рис. fig. 40).

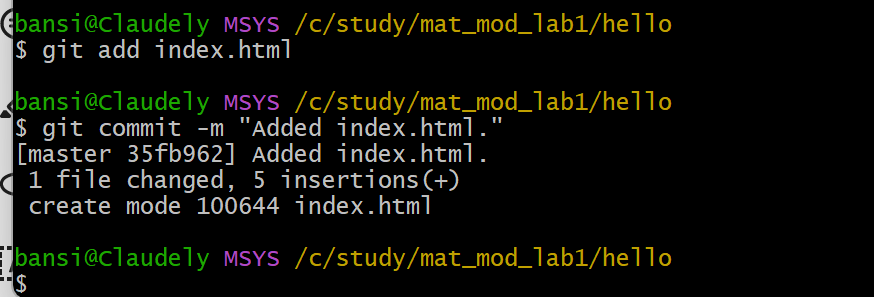


Рис. 40: коммит

Теперь при открытии index.html, увидим кусок страницы hello в маленьком окошке .



Рис. 41: Результат открытия index.html

## 2.17 Git внутри: Каталог .git

### 2.17.1 Каталог .git

ls -C .git Это каталог, в котором хранится вся информация git (рис. fig. 42).

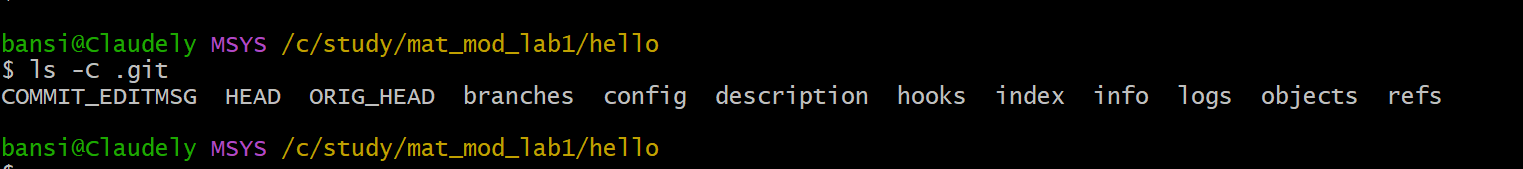


Рис. 42: информация git

### 2.17.2 База данных объектов

ls -C .git/objects

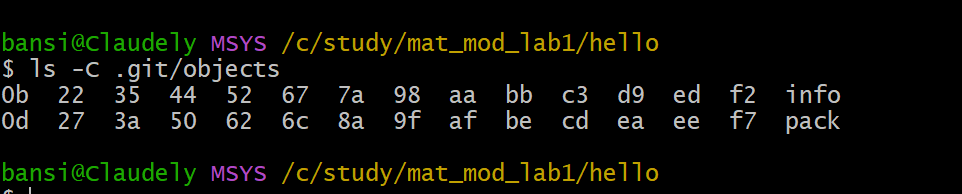


Рис. 43: информация git

### 2.17.3 глубляемся в базу данных объектов

ls -C .git/objects/

Смотрим в один из каталогов с именем из 2 букв. Вы увидите файлы с именами из 38 символов. Это файлы, содержащие объекты, хранящиеся в git. Они сжаты и закодированы, поэтому просмотр их содержимого нам мало чем поможет (рис. fig. 44).



Рис. 44: глубляемся в базу данных объектов

### 2.17.4 Config File

cat .git/config Это файл конфигурации, создающийся для каждого конкретного проекта. Записи в этом файле будут перезаписывать записи в файле .gitconfig вашего главного каталога, по крайней мере в рамках этого проекта (рис. fig. 45).

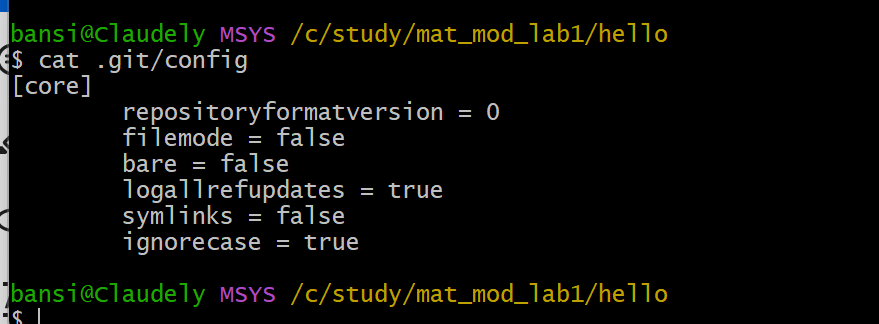


Рис. 45: Config File

### 2.17.5 Ветки и теги

узнавать файлы в подкаталоге тегов. Каждый файл соответствует тегу, ранее созданному с помощью команды git tagЕго содержание — это всего лишь хэш коммита, привязанный к тегу(рис. fig. 46).

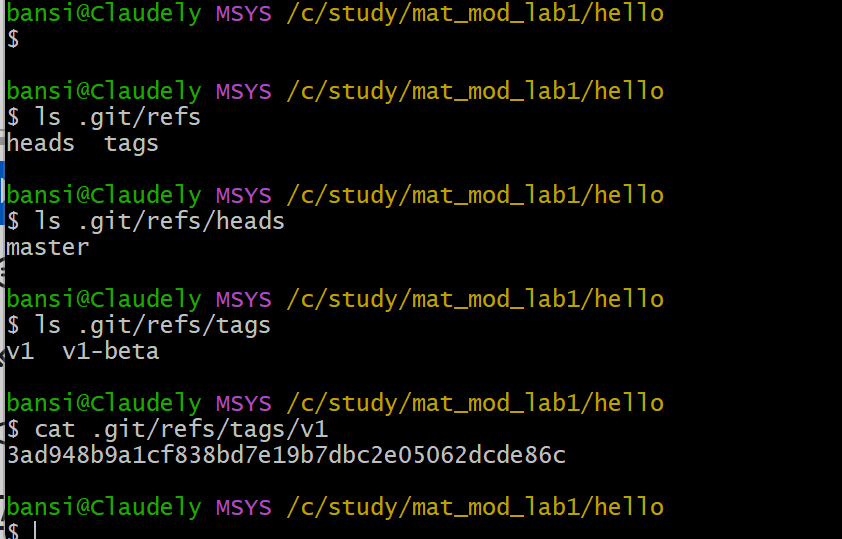


Рис. 46: Ветки и теги

### 2.17.6 Файл HEAD

cat .git/HEAD Файл HEAD содержит ссылку на текущую ветку, в данный момент это должна быть ветка master (рис. fig. 47)

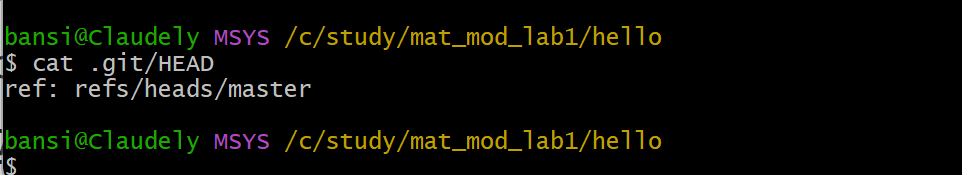


Рис. 47: Файл HEAD

## 2.18 Работа непосредственно с объектами git

Найдем последний коммит и выедем его с помощью SHA1 хэша. Затем посмотрим дерево каталогов, ссылка на который идёт в последнем коммите, вывдем каталог lib и файл hello.html(рис. fig. 47).

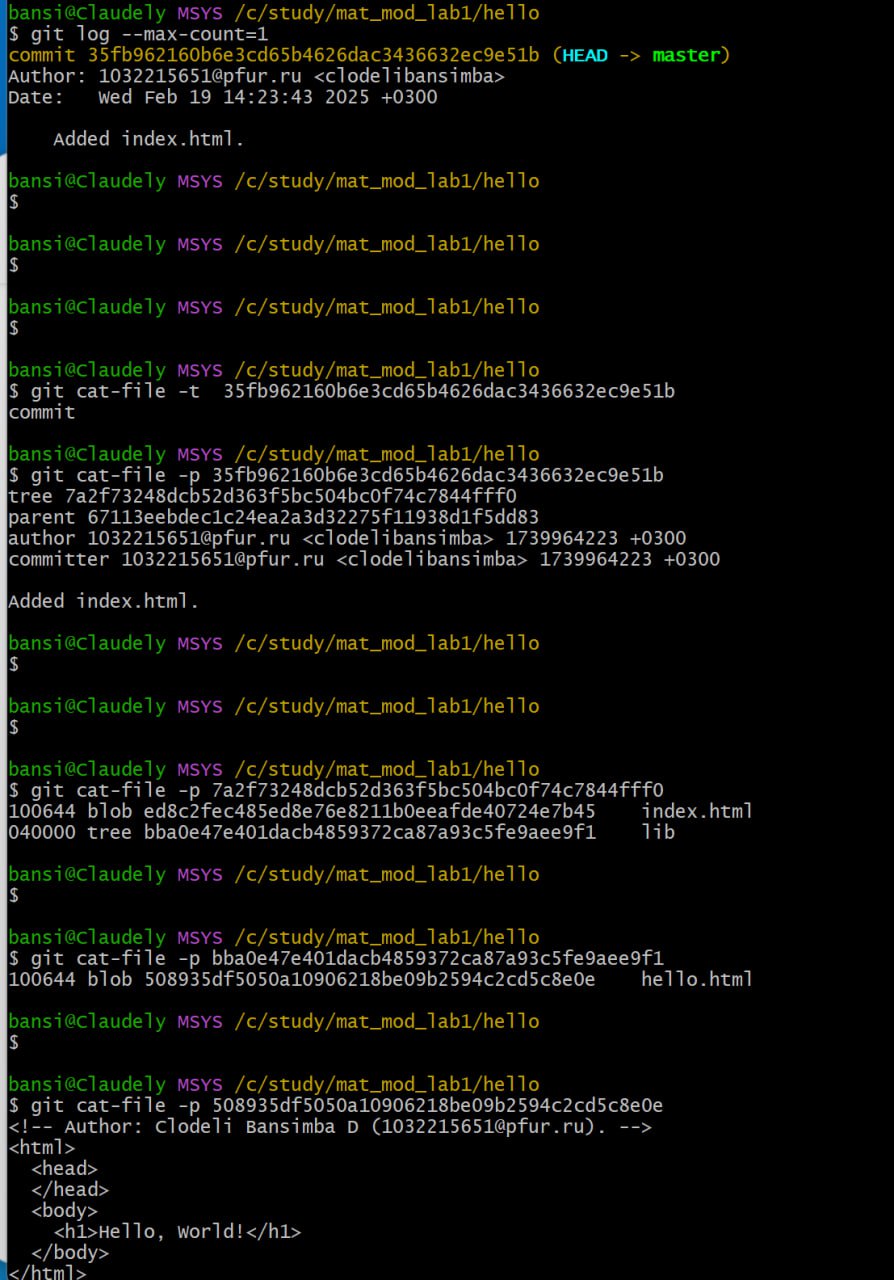


Рис. 48: Работа непосредственно с объектами git

Исследуем git репозиторий вручную самостоятельно. Используя хэш родительского коммита последовательно дойдем до первой версии файла hello.html и посмотрим его.

## 2.19 Создание ветки

Создадим новую ветку «style» и перейдем в неё. Добавим туда файл стилей style.css и добавим его в репозиторий. Обновим файл hello.html, чтобы использовать стили style.css и index.html, также обновим их в репозиторий .

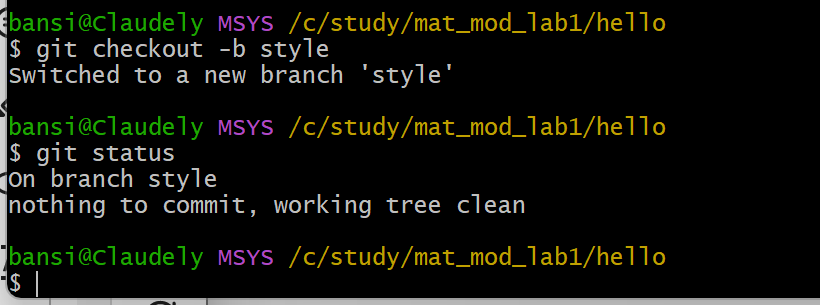


Рис. 49: Создание ветки

добавим файл стилей style.css

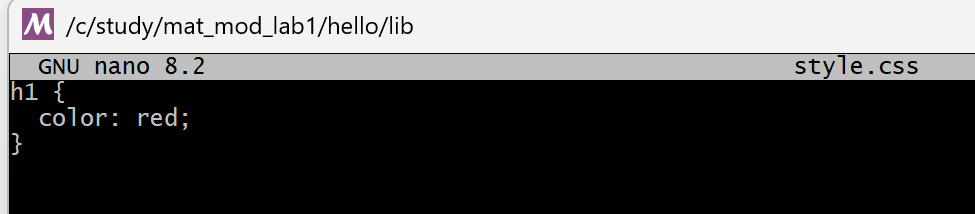


Рис. 50: добавим файл стилей

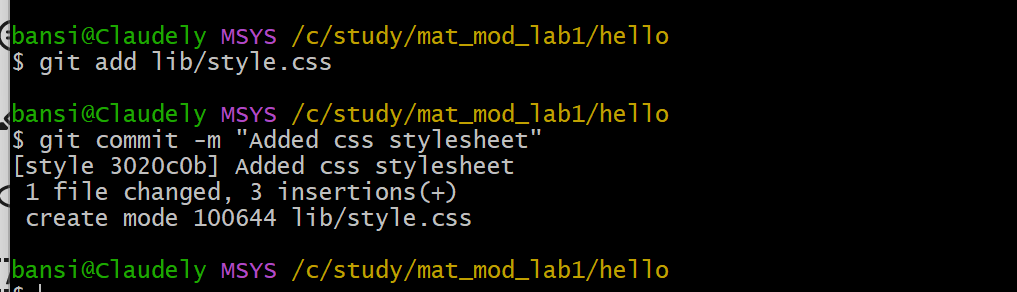


Рис. 51: добавим файл стилей

Обновим файл hello.html

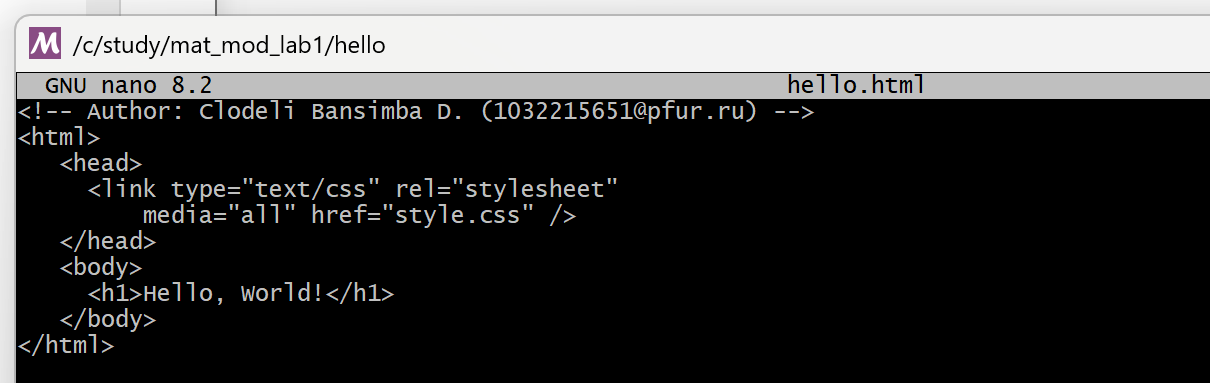


Рис. 52: Редактирование файла

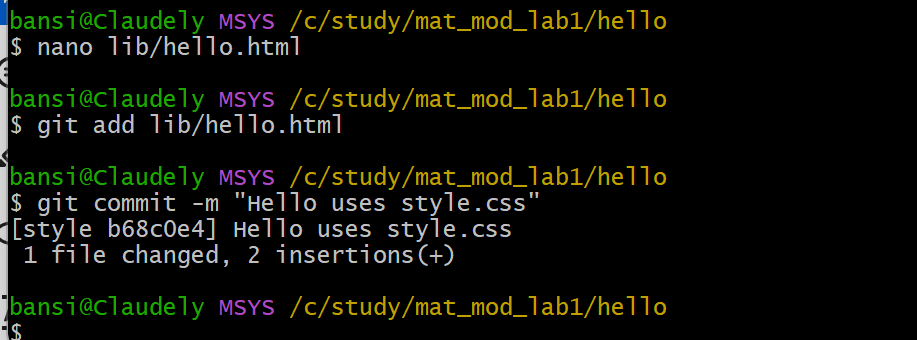


Рис. 53: Редактирование файла

## 2.20 Навигация по веткам

Посмотрим все логи(рис. fig. 54).

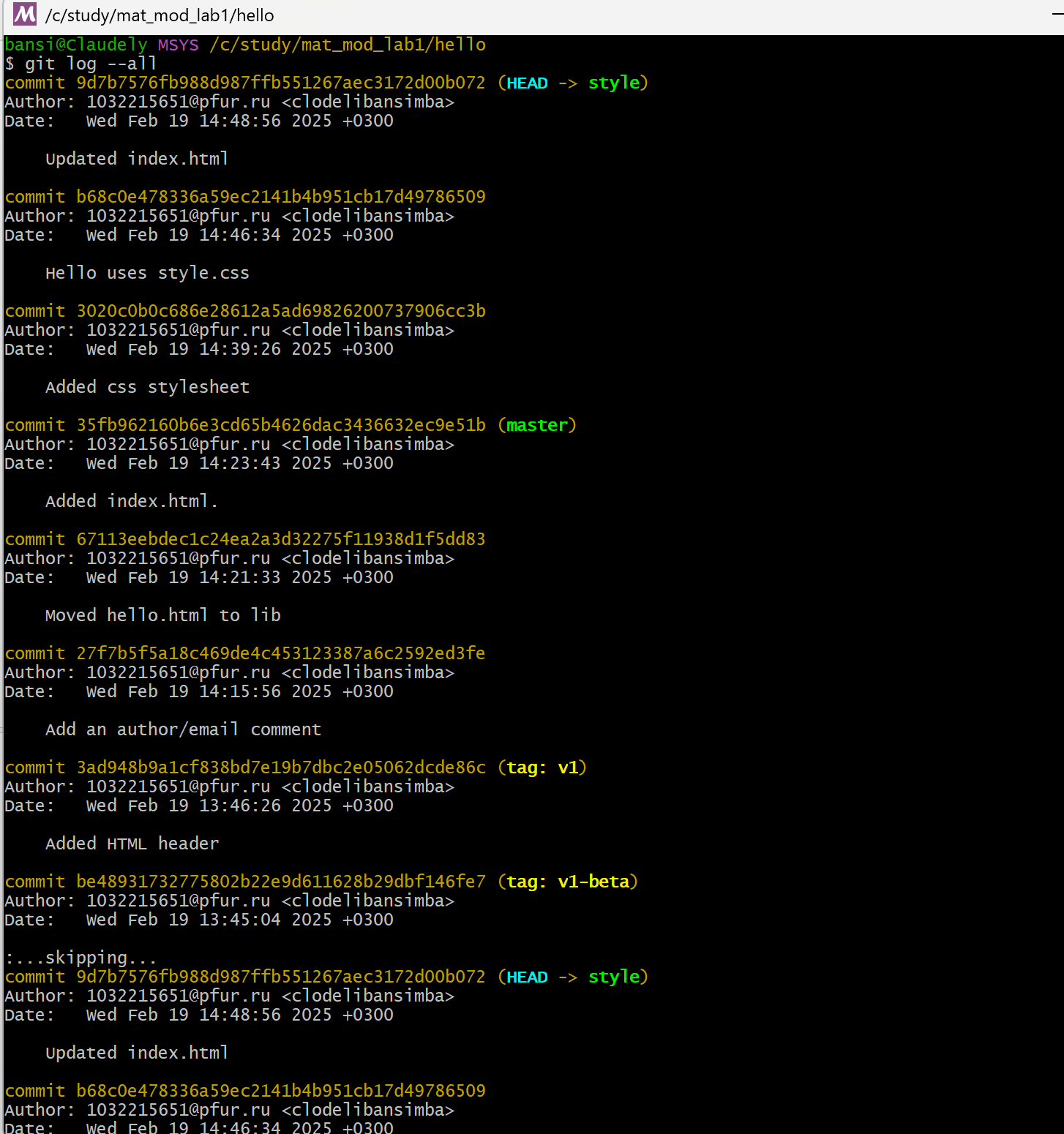


Рис. 54: Просмотр логов новой ветки

Переключимся обратно на основную ветку и просмотрим содержимое файла ib/hello.html, заметим, что он не использует стили, также просмотрим содержимое этого файла в новой ветке(рис. fig. 5556).

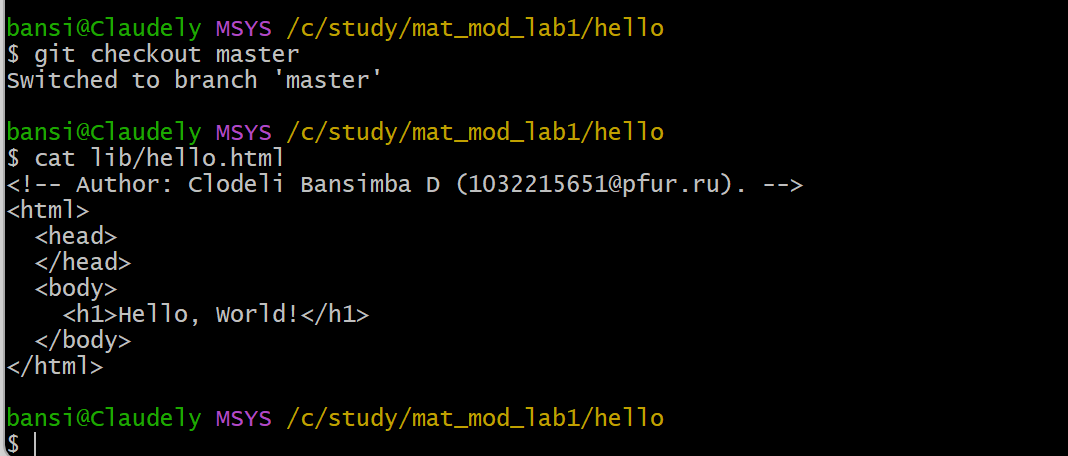


Рис. 55: Переключение на ветку master

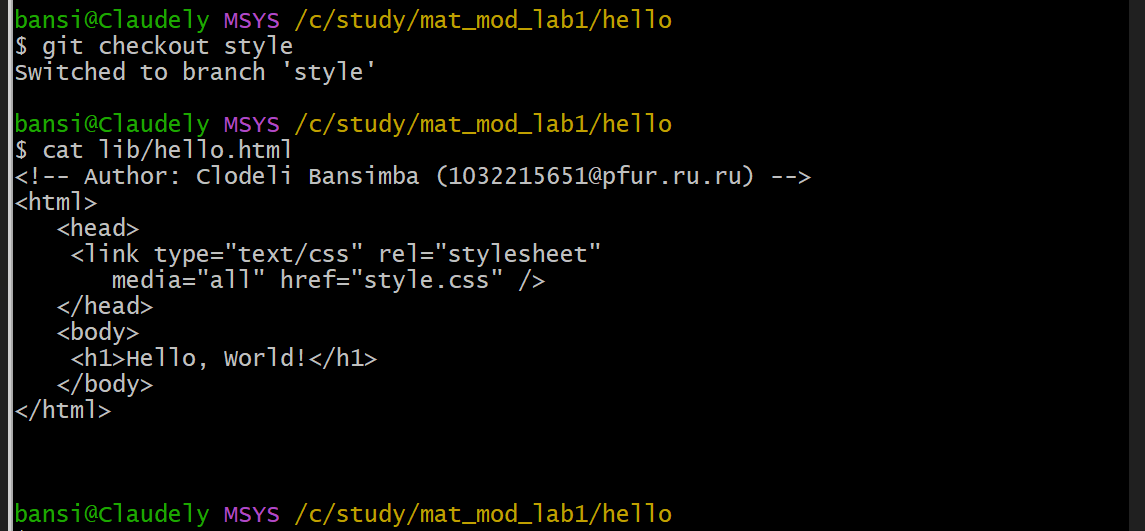


Рис. 56: Возвращение к ветке style

## 2.21 Изменения в ветке master

Вернемся в основную ветку и добавим файл README.md. Просмотрим ветки и их различия(рис. fig. 57).

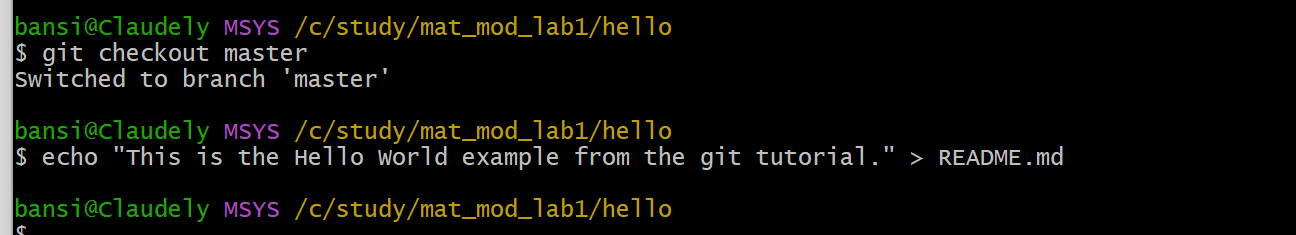


Рис. 57: Изменения в ветке master

## 2.22 Просмотрите текущие ветки

Сделайте коммит изменений README.md в ветку master.

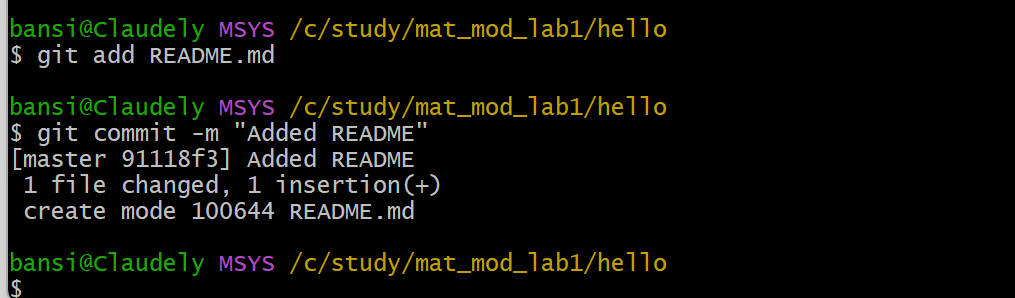


Рис. 58: коммит

для просмотра веток и их отличий.

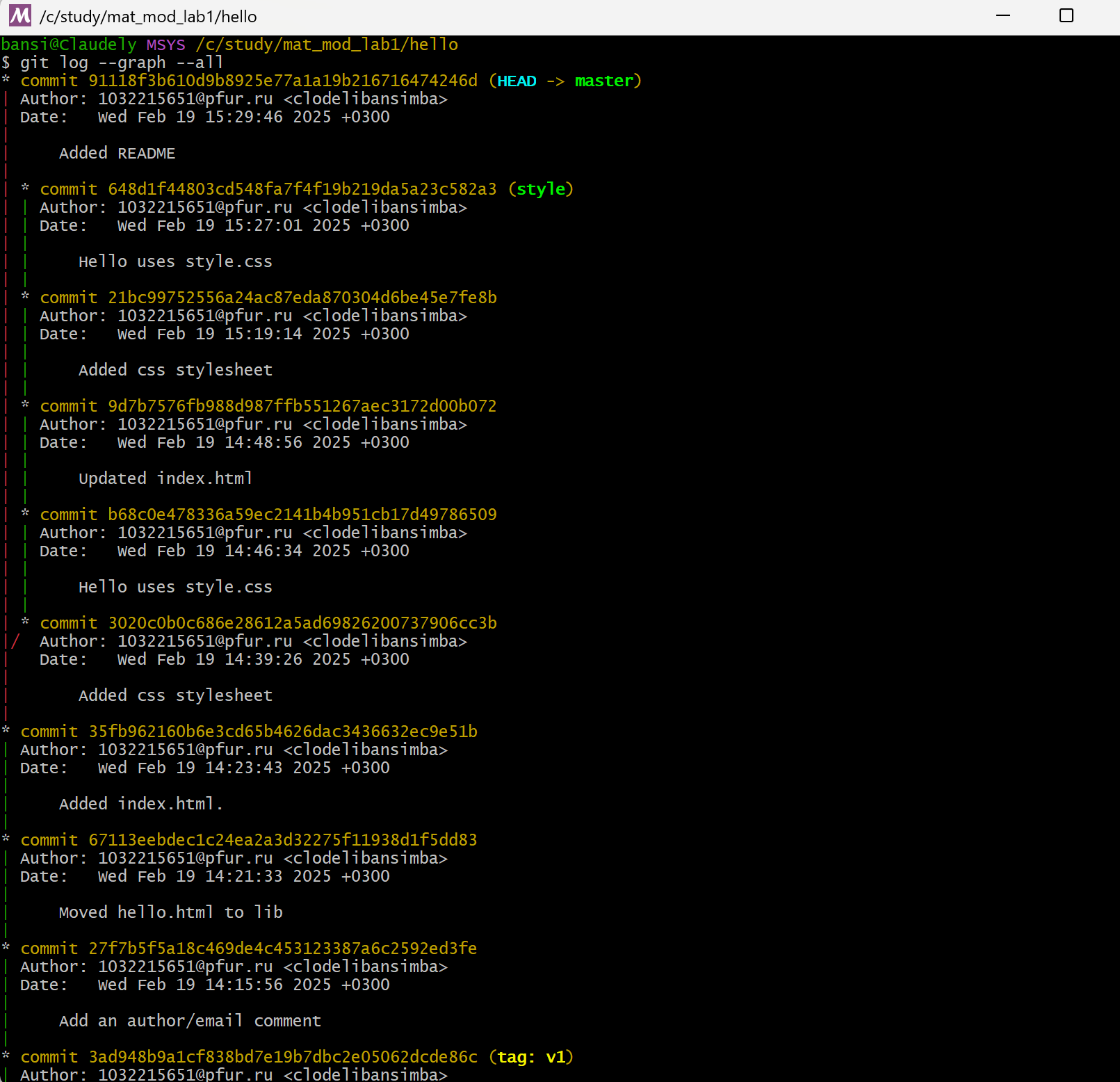


Рис. 59: для просмотра веток и их отличий

## 2.23 Слияние

Слияние переносит изменения из двух веток в одну. Вернемся к ветке style и сольем master с style(рис. fig. 60).

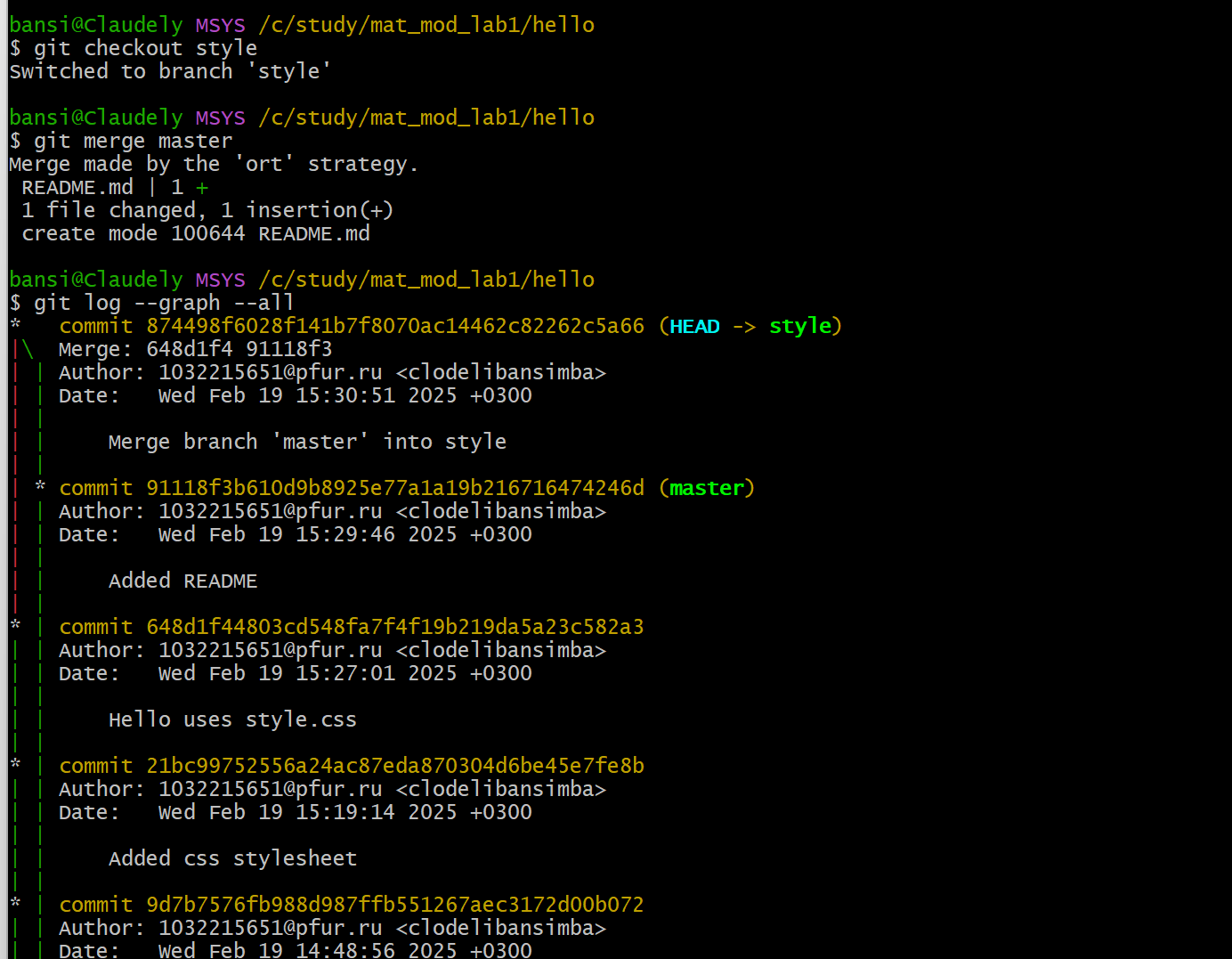


Рис. 60: Слияние веток

## 2.24 Создание конфликта

Вернемся в ветку master и создадим конфликт, внеся изменения в файл hello.html (рис. fig. 61).

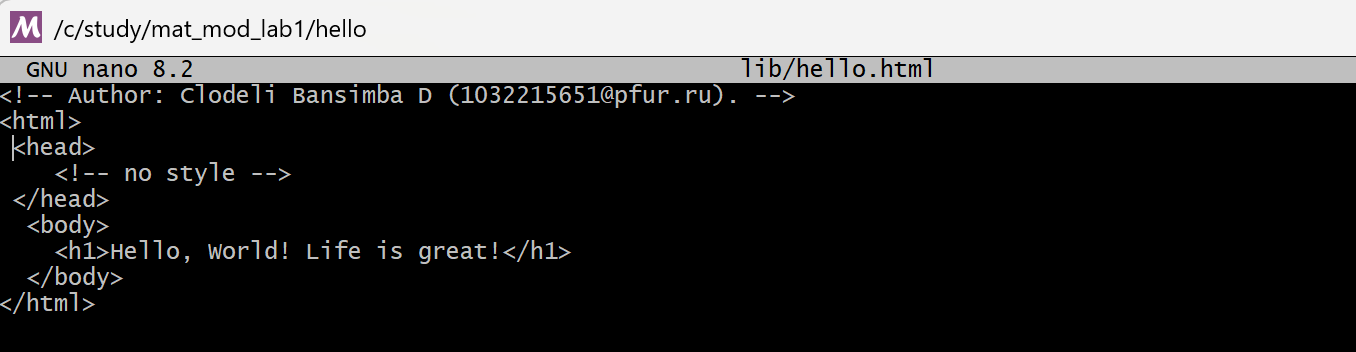


Рис. 61: Создание конфликта

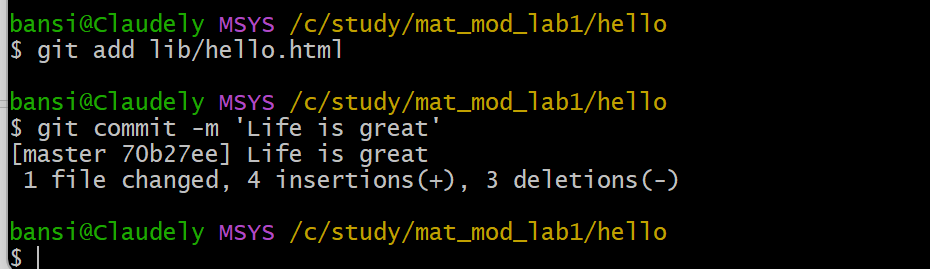


Рис. 62: Создание конфликта

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. fig. 63).

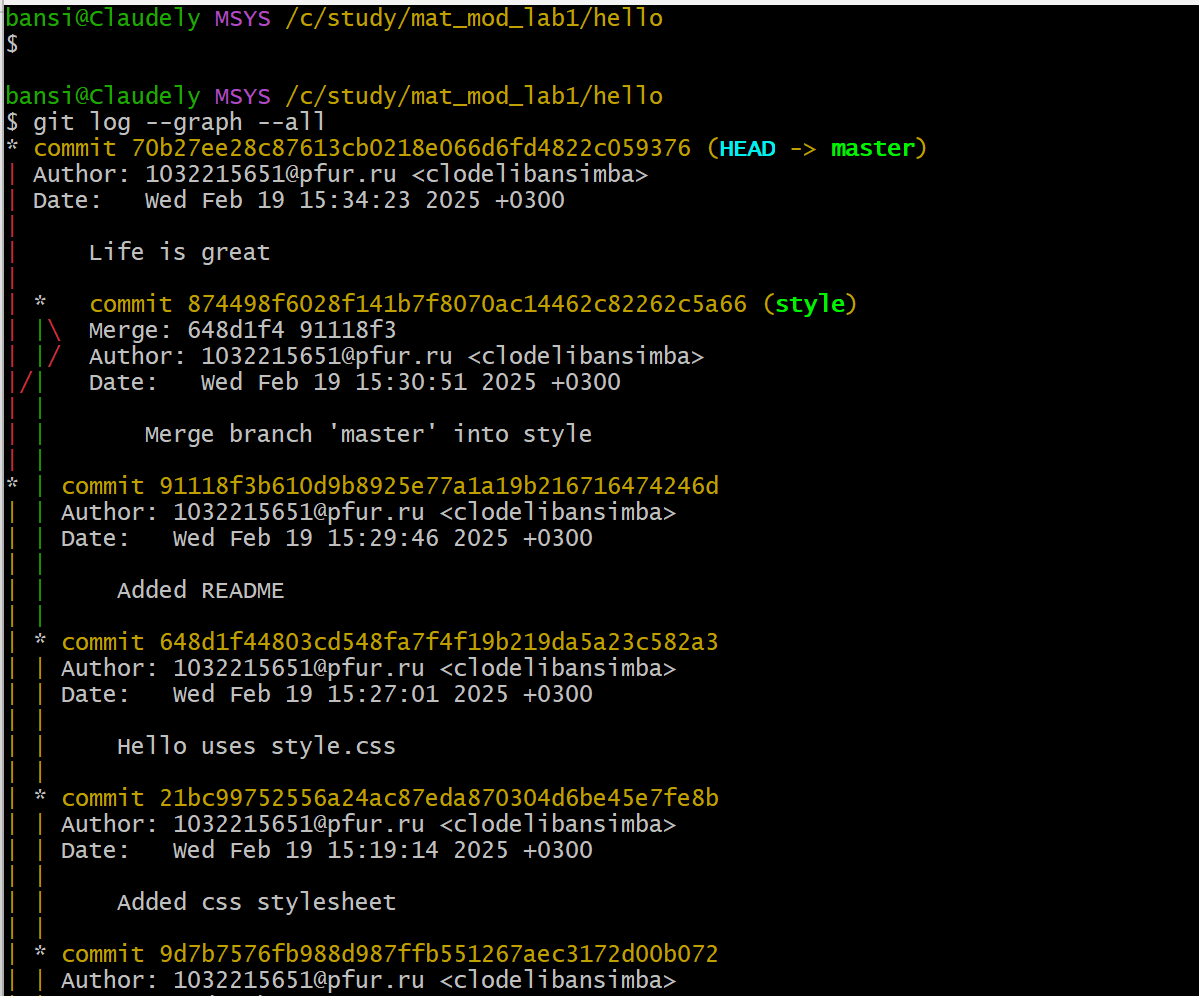


Рис. 63: Просмотр веток

## 2.25 Разрешение конфликтов

### 2.25.1 Слияние master с веткой style

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 (рис. fig. 64) .

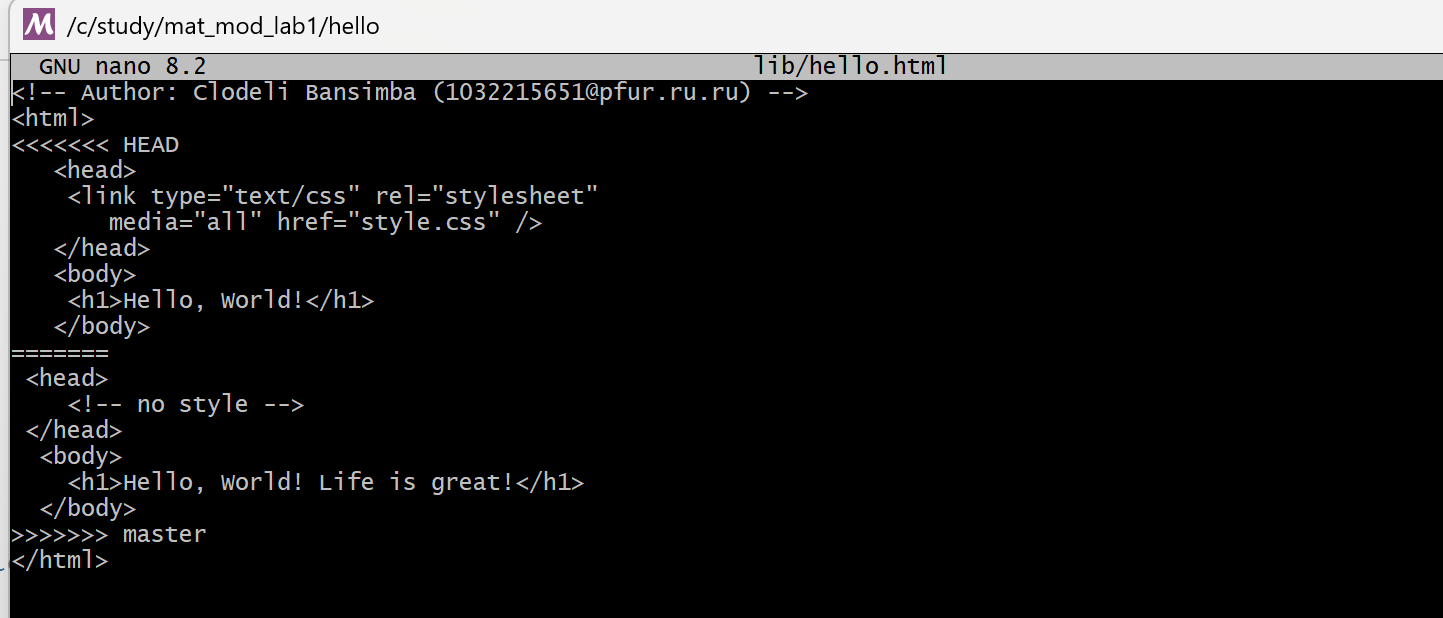


Рис. 64: Конфликт

Внесем изменения в lib/hello.html, оставив только необходимую нам запись и добавим этот файл в репозиторий, чтобы вручную разрешить конфликт (рис. fig. 65).

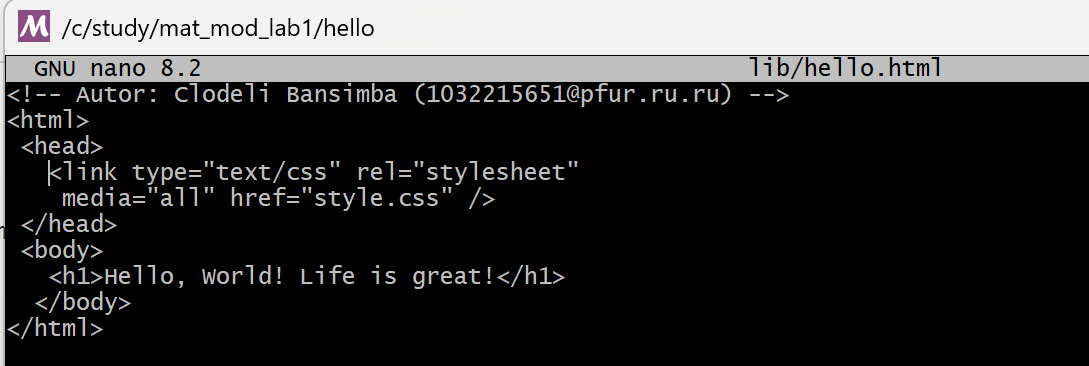


Рис. 65: Конфликт

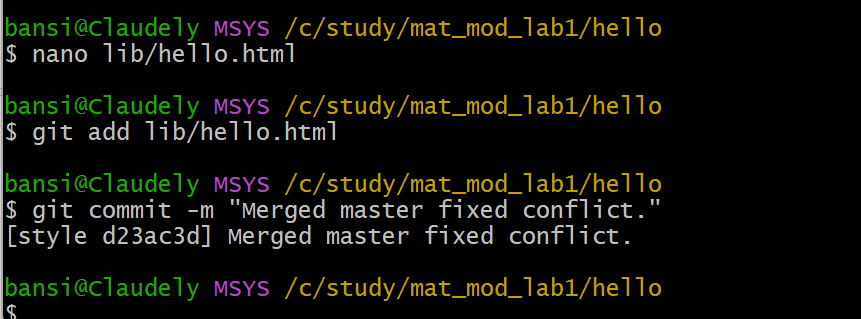


Рис. 66: Разрешение конфликта

## 2.26 Сброс ветки style

Вернемся на ветке 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. fig. 67).

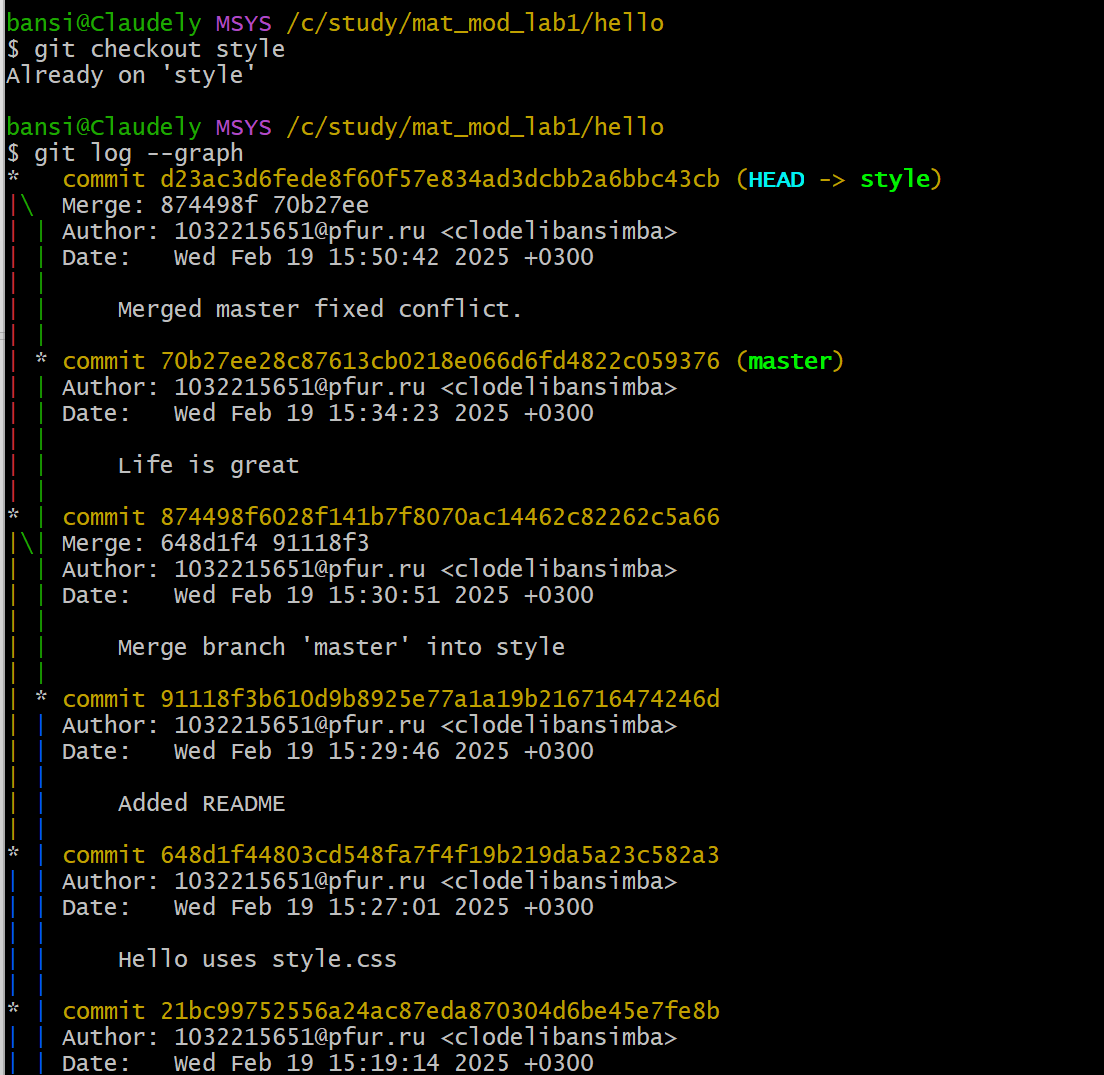


Рис. 67: Поиск коммита перед слиянием

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

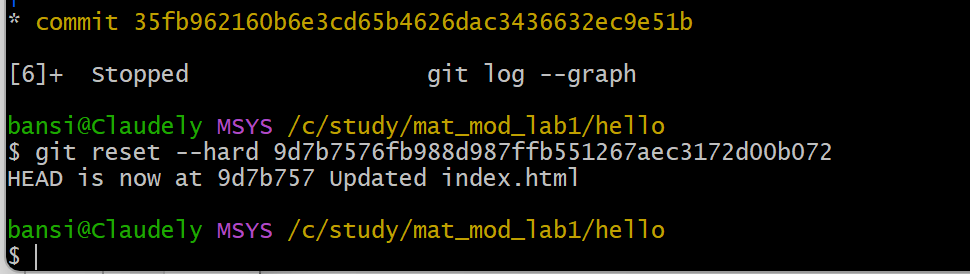


Рис. 68: Поиск коммита перед слиянием

Поищим лог ветки style. Увидим, что у нас в истории больше нет коммитов слияний(рис. fig. 69).

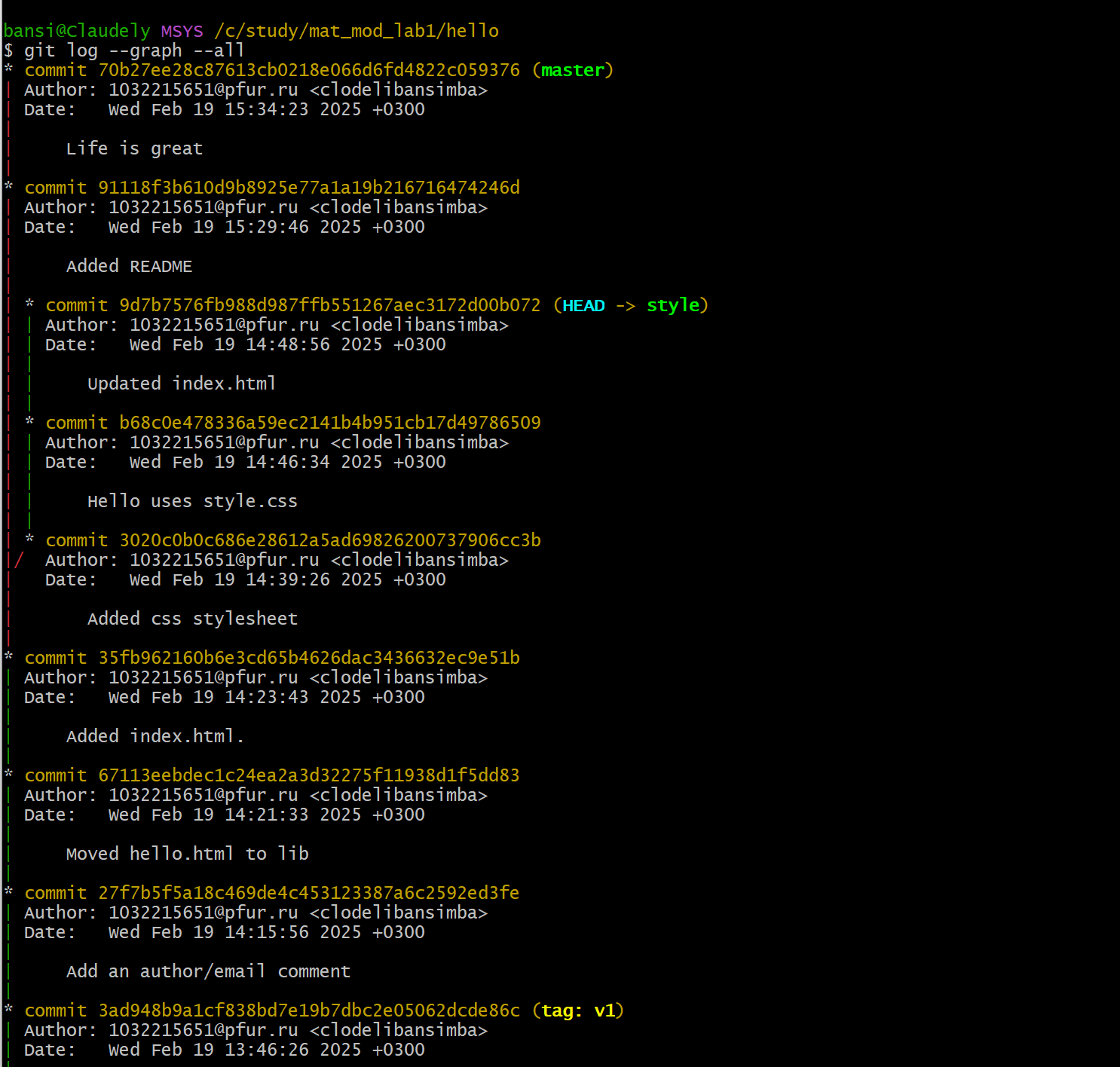


Рис. 69: Сброс ветки style

## 2.27 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. fig. 70).

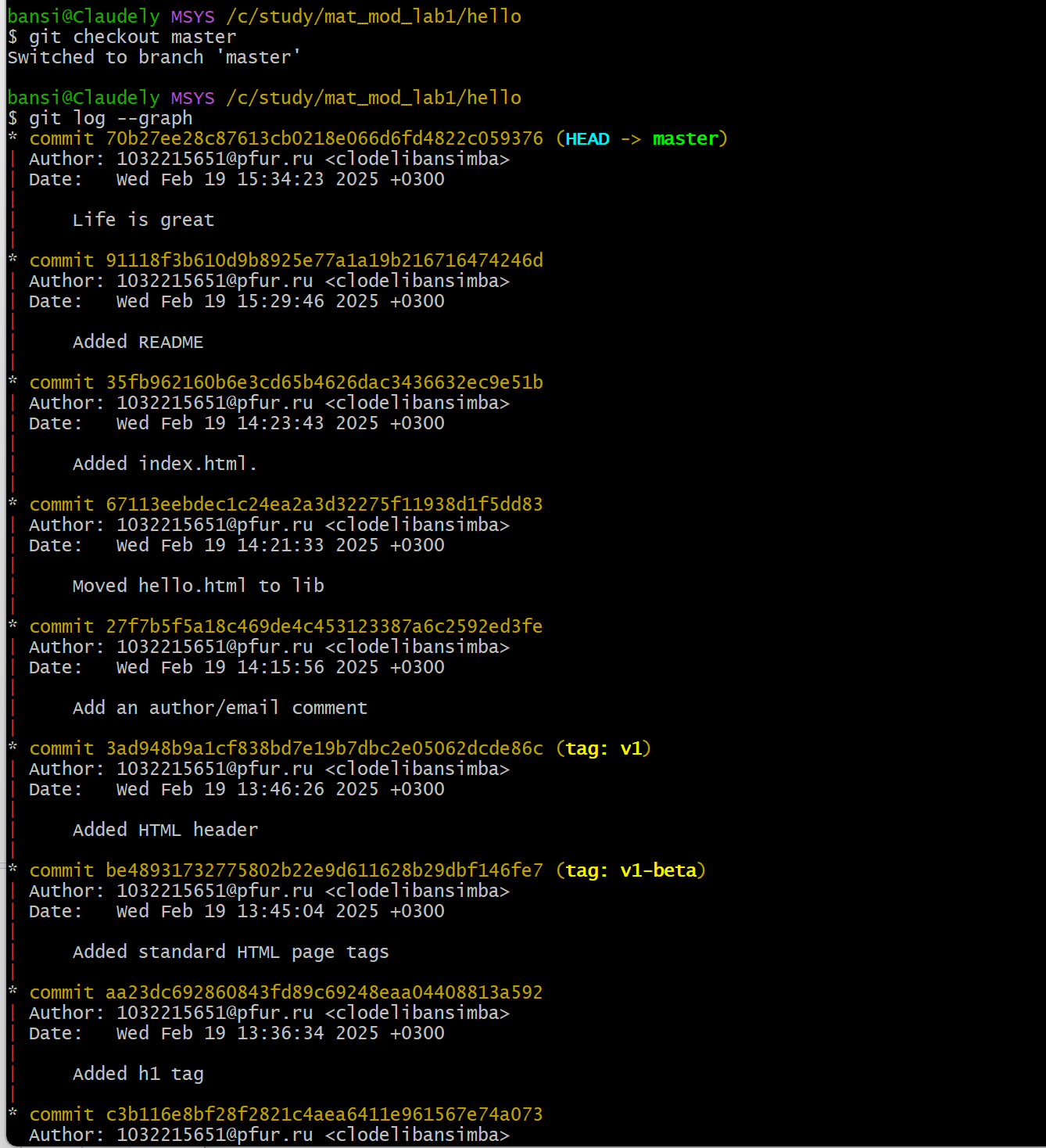


Рис. 70: Поиск коммита перед конфликтом

Коммит «Added README» идет непосредственно перед коммитом конфликтующего интерактивного режима. Мы сбросим ветку master к коммиту «Added README»(рис. fig. 71).



Рис. 71: Сброс ветки master

## 2.28 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. fig. 72).

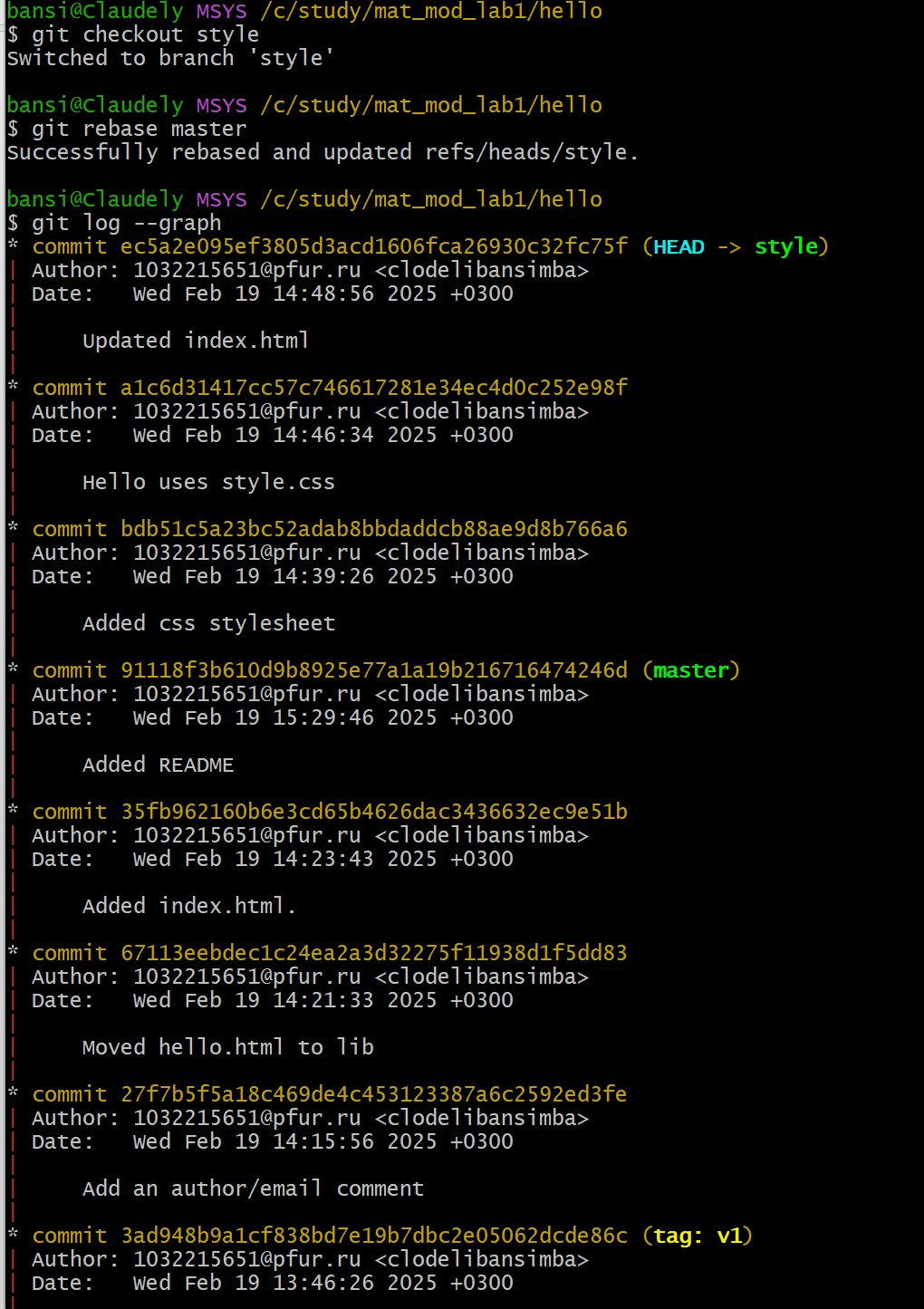


Рис. 72: Перебазирование

## 2.29 Слияние в ветку master

Вернемся в ветку master и сольем ветку style в неё с помощью команды git merge(рис. fig. 73).

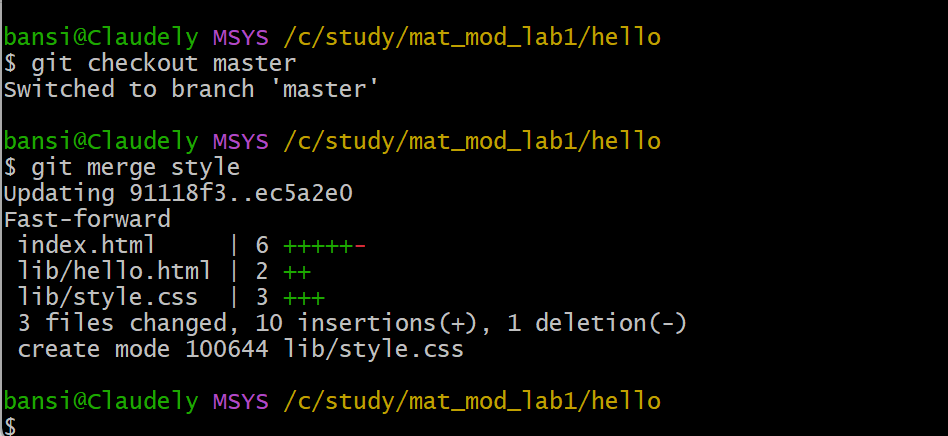


Рис. 73: Слияние style в master

Теперь ветки style и master идентичны (рис. fig. 74).



Рис. 74: Слияние style в master

## 2.30 Клонирование репозиториев

Перейдем в наш рабочий каталог и сделаем клон репозитория hello, затем создадим клон репозитория. Просмотрев его увидим список всех файлов на верхнем уровне оригинального репозитория README.md, index.html и lib. Затем просмотрим историю репозитория и увидим список всех коммитов в новый репозиторий, и он совпадает с историей коммитов в оригинальном репозитории. Единствен в названиях веток (рис. fig. 75) и (рис. fig. 76).

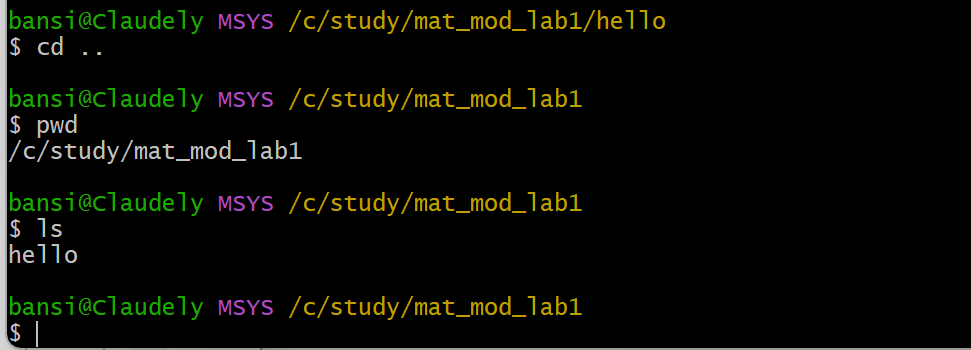


Рис. 75: Клонирование репозиториев

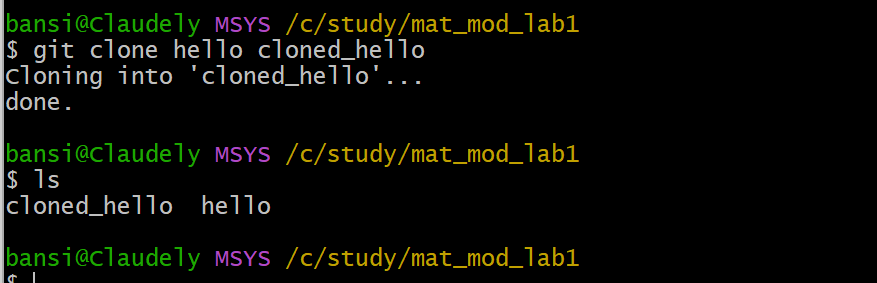


Рис. 76: Клонирование репозиториев

## 2.31 Что такое origin?

Клонированный репозиторий знает об имени по умолчанию удаленного репозитория. Посмотрим, подробную информацию об имени по умолчанию.Для того, чтобы увидеть все ветки используем опцию -a (рис. fig. 77).

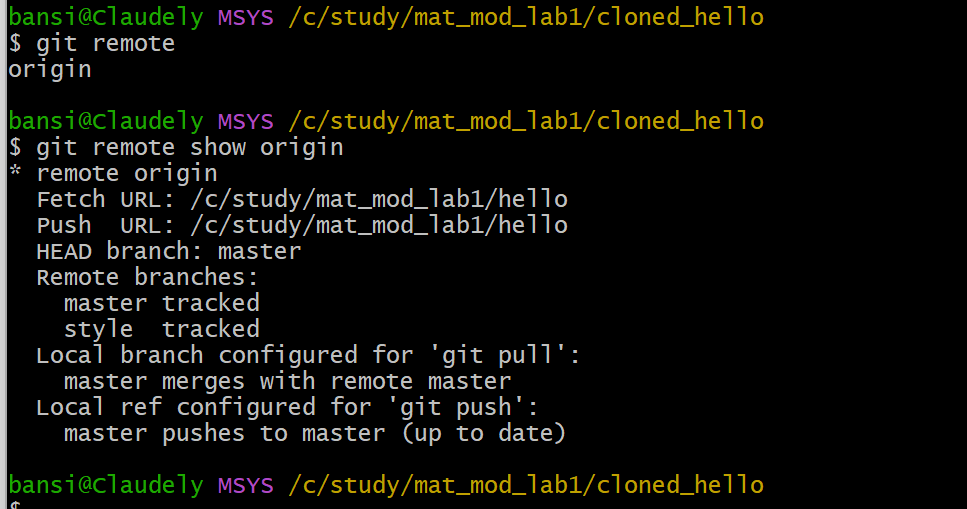


Рис. 77: Просмотр имени по умолчанию удаленного репозитория

## 2.32 Удаленные ветки

Посмотрим на ветки, доступные в нашем клонированном репозитории. Можно увидеть, что в списке только ветка master (рис. fig. 78).

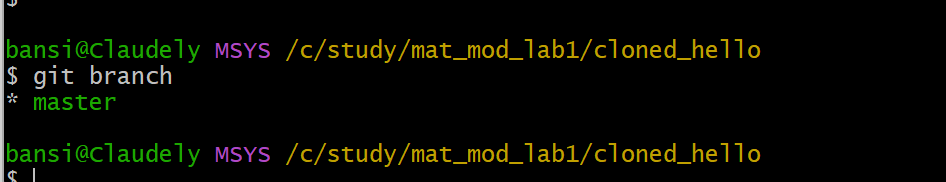


Рис. 78: Удаленные ветки

Для того, чтобы увидеть все ветки (рис. fig. 79)

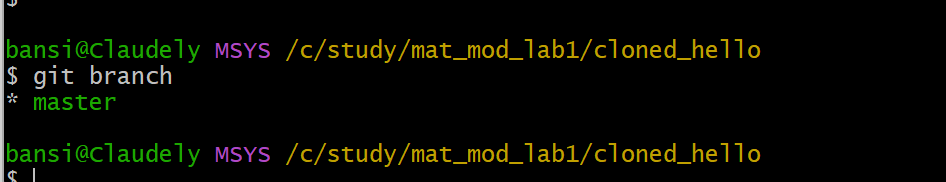


Рис. 79: Удаленные ветки

## 2.33 Изменение оригинального репозитория

Перейдем в репозиторий hello. Внесем изменения в файл README.md. Затем добавим их в репозиторий (рис. fig. 80).

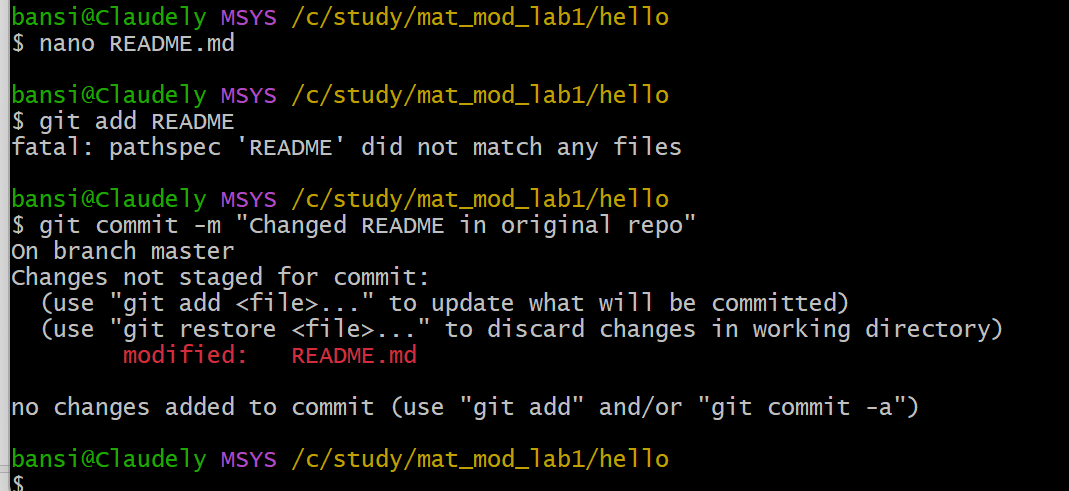


Рис. 80: Изменение оригинального репозитория

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. fig. 81).

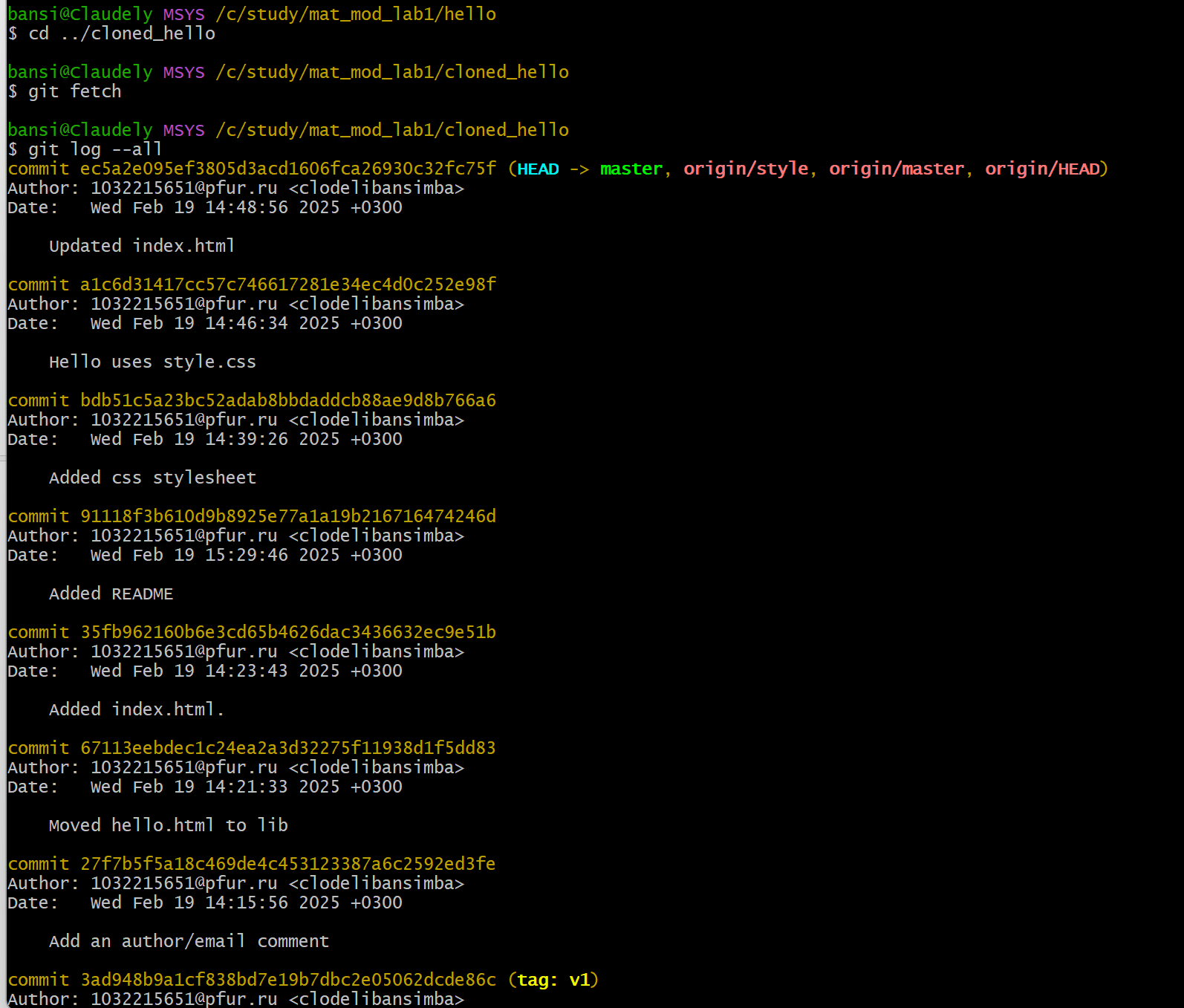


Рис. 81: Извлечение изменений

Мы можем продемонстрировать, что клонированный файл README.md не изменился (рис. fig. 82).

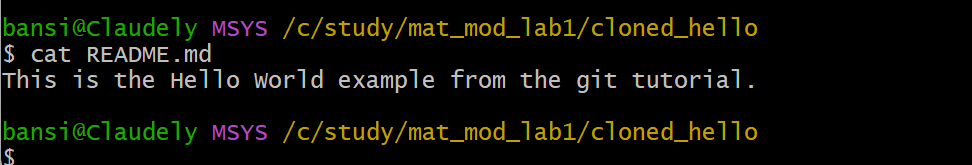


Рис. 82: Извлечение изменений

## 2.34 Слияние извлеченных изменений

Сольем внесённые изменения в главную ветку.

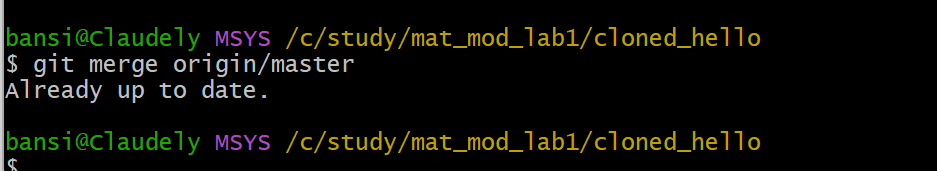


Рис. 83: Слияние извлеченных изменений

Также можно было бы использовать команду git pull, которая является объединением fetch и merge в одну команду (рис. fig. 84), (рис. fig. 85).

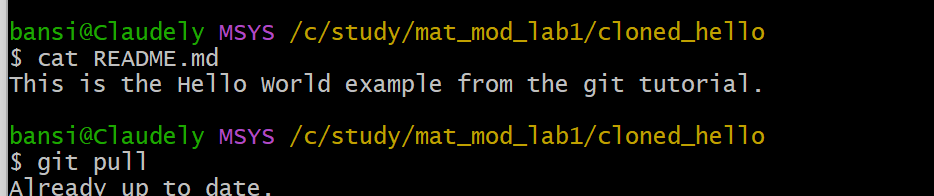


Рис. 84: Слияние извлеченных изменений

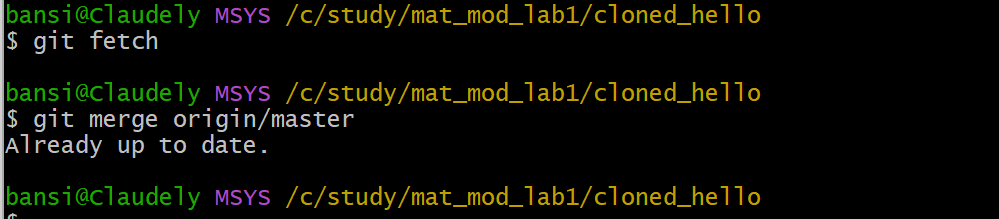


Рис. 85: Слияние извлеченных изменений

## 2.35 Добавление ветки наблюдения

Добавим локальную ветку, которая отслеживает удаленную ветку, теперь мы можем видеть ветку style в списке веток и логе(рис. fig. 86).

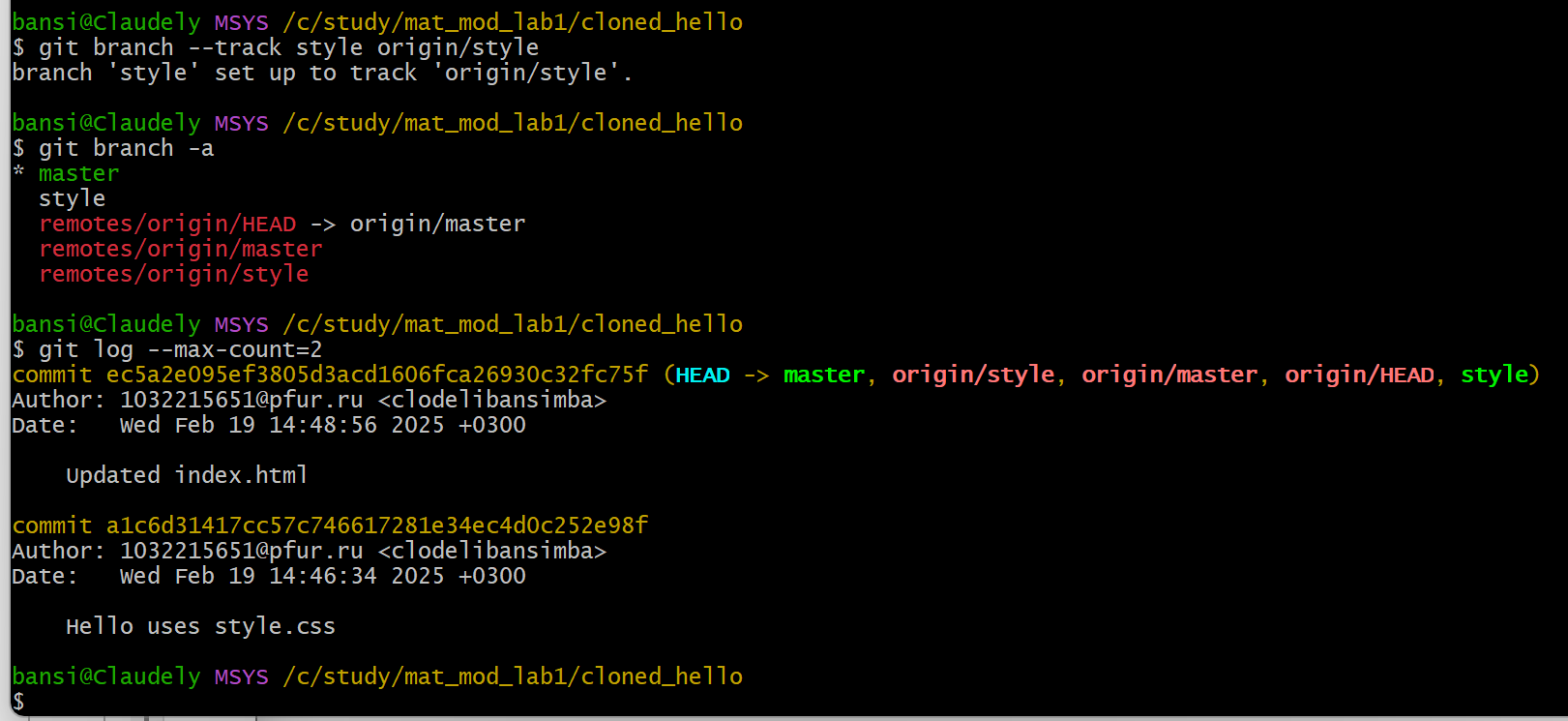


Рис. 86: Добавление ветки наблюдения

## 2.36 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. fig. 87).

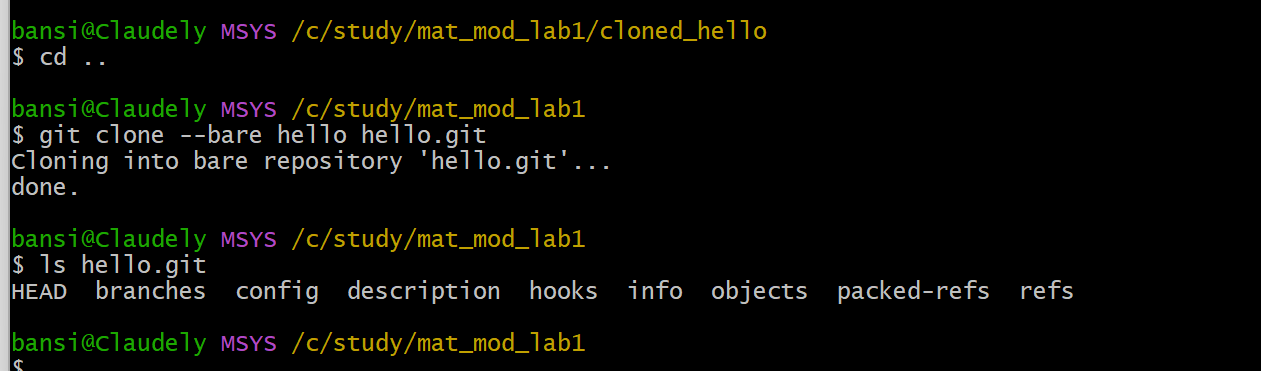


Рис. 87: Создание чистого репозитория

## 2.37 Добавление удаленного репозитория

добавим репозиторий hello.git к нашему оригинальному репозиторию (рис. fig. 88).

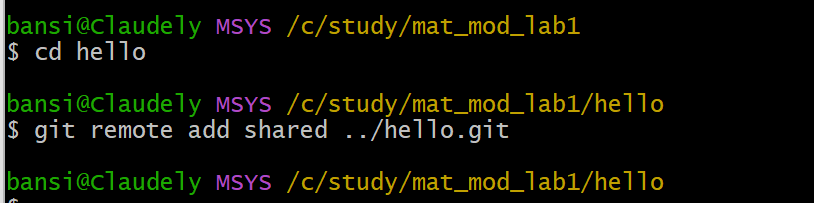


Рис. 88: добавим репозиторий hello.git

## 2.38 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 (рис. fig. 89, fig. 90).

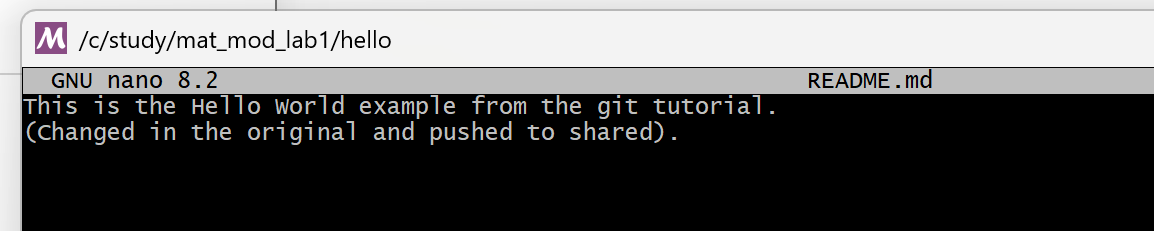


Рис. 89: Отправка изменений

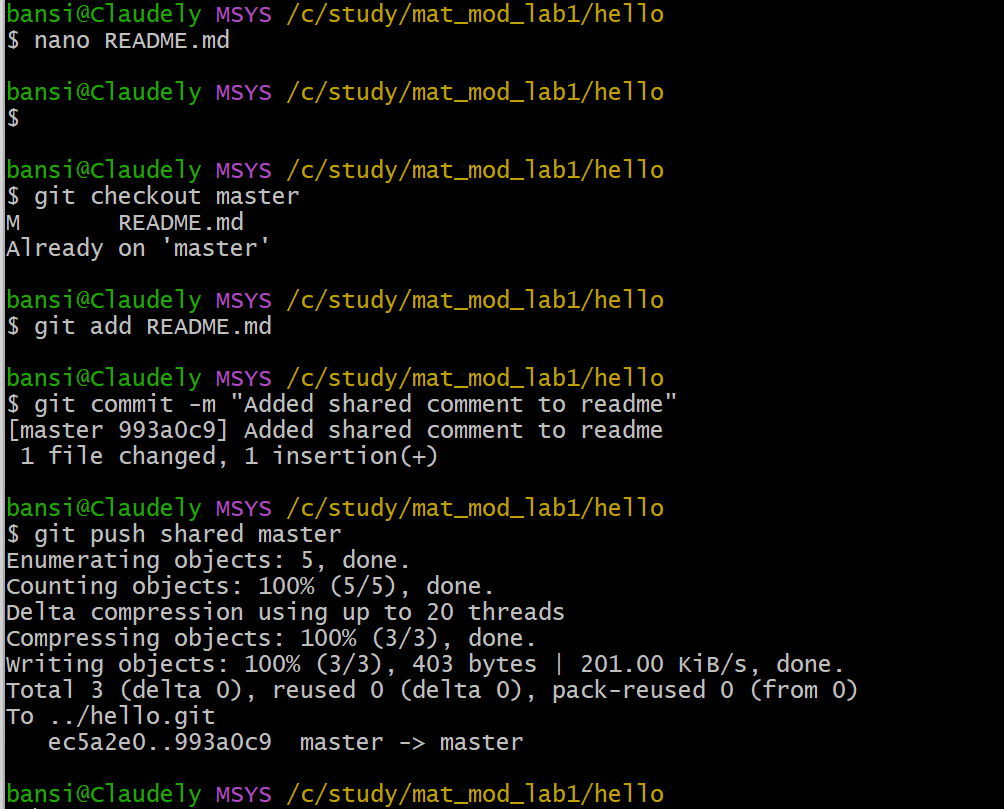


Рис. 90: Отправка изменений

Затем извлечем изменения из общего репозитория (рис. fig. 90).

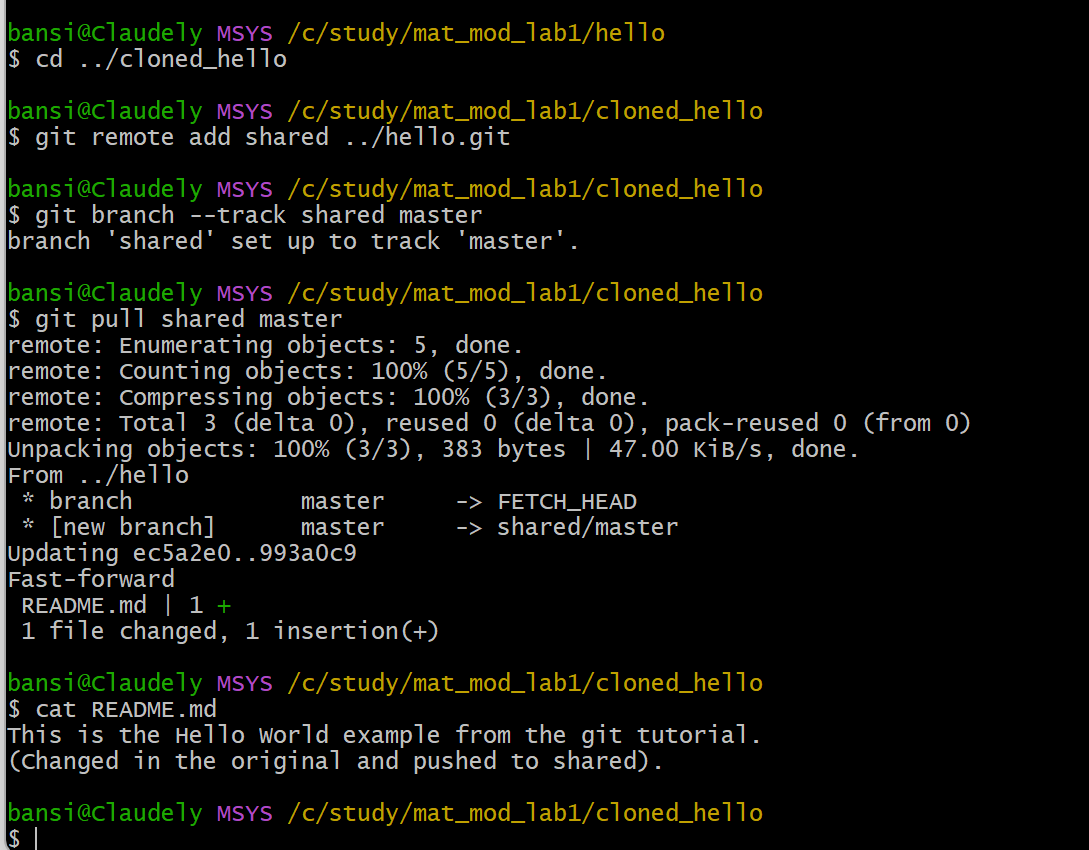


Рис. 91: Извлечение изменений

# 3 Выводы

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

# Список литературы