Отчёт по лабораторной работе №1

Работа с git

Жукова Виктория Юрьевна"

Содержание"

# Цель работы

Цель данной работы состоит в том, чтобы освежить знания git и познакомиться с markdown.

# Задание

Сделать отчёт по предыдущей лабораторной работе в формате Markdown.

В качестве отчёта просьба предоставить отчёты в 3 форматах:  
pdf, docx и md (в архиве, поскольку он должен содержать скриншоты, Makefile и т.д.)

# Теоретическое введение

## Определение

Git — распределённая система контроля версий, которая даёт возможность разработчикам отслеживать изменения в файлах и работать над одним проектом совместно с коллегами. Git известен своей скоростью, простым дизайном, поддержкой нелинейной разработки, полной децентрализацией и возможностью эффективно работать с большими проектами.

Подход Git к хранению данных похож на набор снимков миниатюрной файловой системы. Каждый раз, когда вы сохраняете состояние своего проекта в Git, система запоминает, как выглядит каждый файл в этот момент, и сохраняет ссылку на этот снимок.

## История

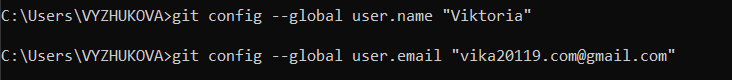
Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux, первая версия выпущена 7 апреля 2005 года. На сегодняшний день данной системой пользуются миллионы разработчиков по всему миру.

# Выполнение лабораторной работы

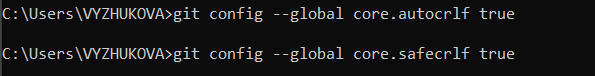
## 1.1 Подготовка

### 1.1.1 Установка имени и электронной почты

Git на моем ноутбуке был уже установлен, я сразу перешла к установке имени и email'а.



### 1.1.2 Параметры установки окончаний строк

Я установила параметры окончания строк с помощью команд  


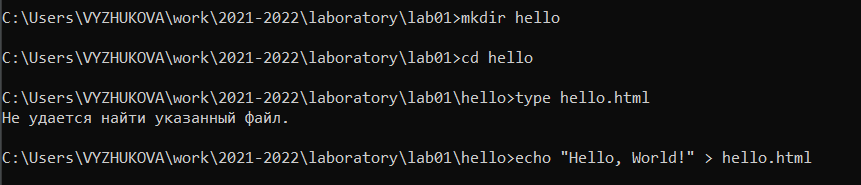
### 1.1.3 Установка отображения unicode

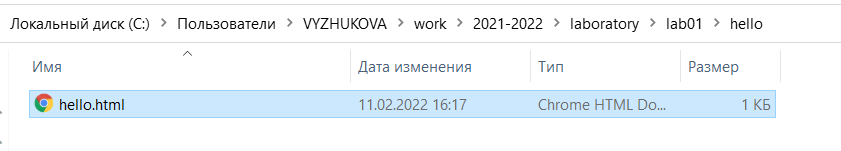
По умолчанию, git будет печатать не-ASCII символов в именах файлов в виде  
восьмеричных последовательностей \nnn. Что бы избежать нечитаемых строк,  
установливаю соответствующий флаг.  
 fig:

## 1.2 Создание проекта

### 1.2.1 Создаю страницу «Hello, World»

Я начнаю работу в пустом рабочем каталоге с создания пустого каталога с именем  
hello, затем вохожу в него и создаю там файл с именем hello.html.





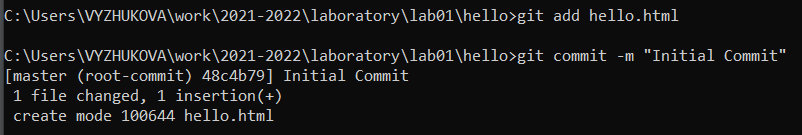
### 1.2.2 Создание репозитория

Чтобы создать git репозиторий из этого каталога, выполняю команду git init.



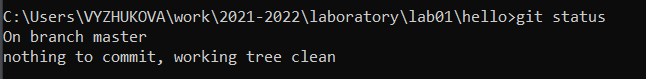
### 1.2.3 Добавление файла в репозиторий

Добавляю файл в репозиторий.



### 1.2.4 Проверка состояние репозитория

Использую команду git status, чтобы проверить текущее состояние репозитория.



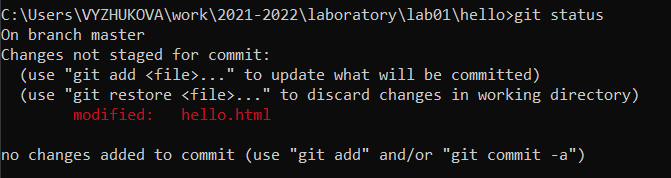
## 1.3 Внесение изменений

### 1.3.1 Измените страницу «Hello, World»

Добавляю кое-какие HTML-теги к нашему приветствию. Изменяю содержимое  
файла hello.html на:

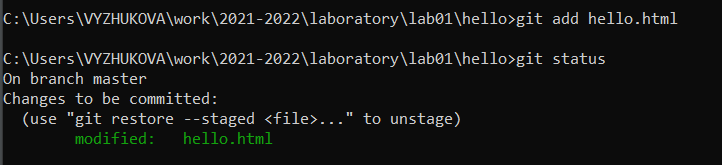


Проверяю состояние рабочего каталога.



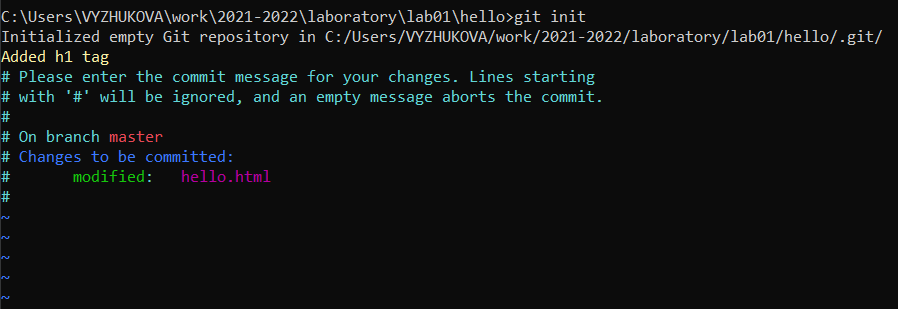
## 1.4 Индексация изменений

Теперь выполняю команду git, чтобы проиндексировать изменения. И проверяю  
состояние.

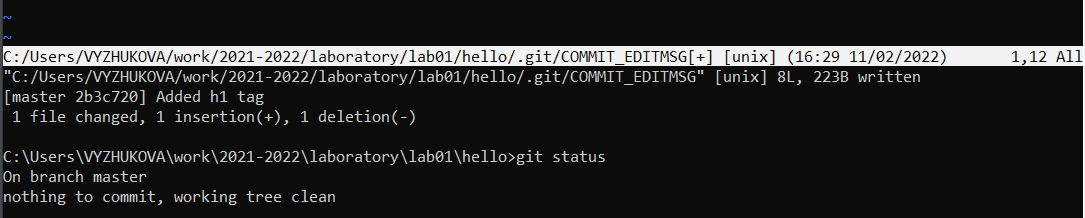


### 1.4.1 Коммит изменений

Делаю коммит с комментарием

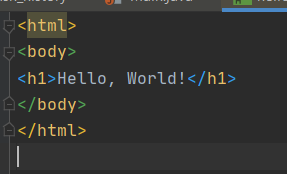


и проверяю состояние.



### 1.4.2 Добавьте стандартные теги страницы

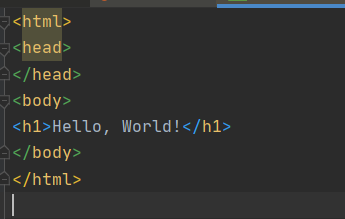
Изменяю страницу «Hello, World», чтобы она содержала стандартные теги <html>  
и <body>.



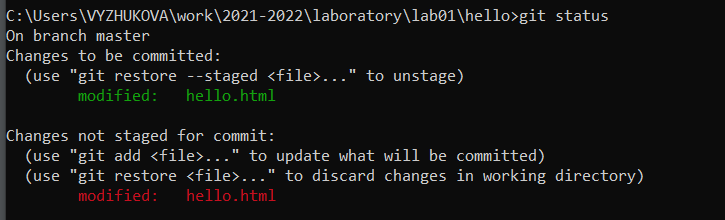
Теперь добавляю это изменение в индекс git.



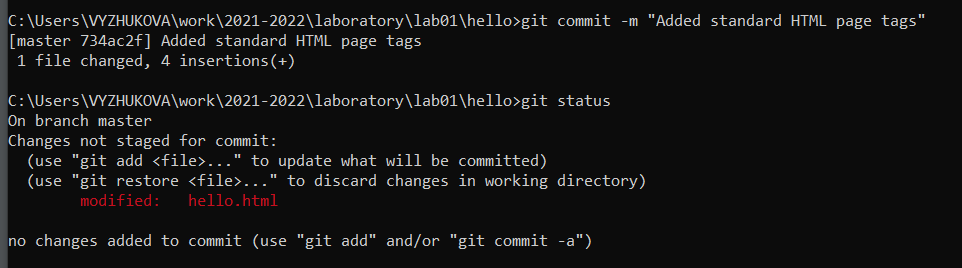
Теперь добавляю заголовки HTML (секцию <head>) к странице «Hello, World».



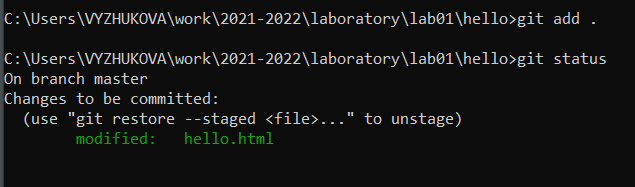
Проверяю текущий статус:



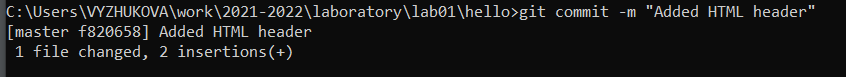
Произвожу коммит проиндексированного изменения (значение по умолчанию), а затем еще раз проверяю состояние.



Теперь добавляю второе изменение в индекс, а затем проверяю состояние с  
помощью команды git status.

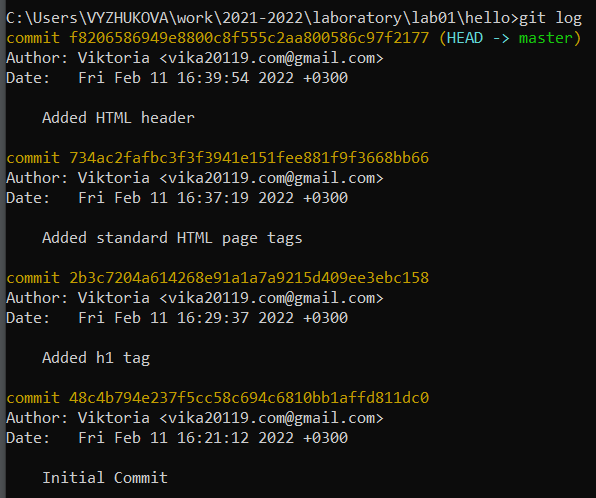
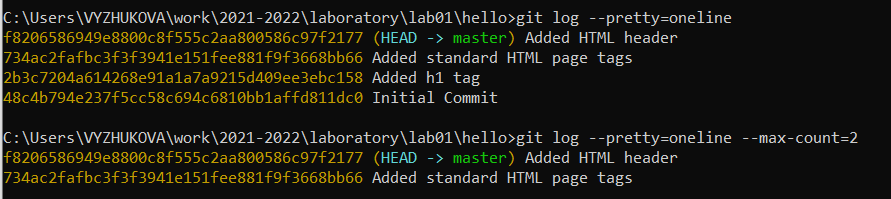
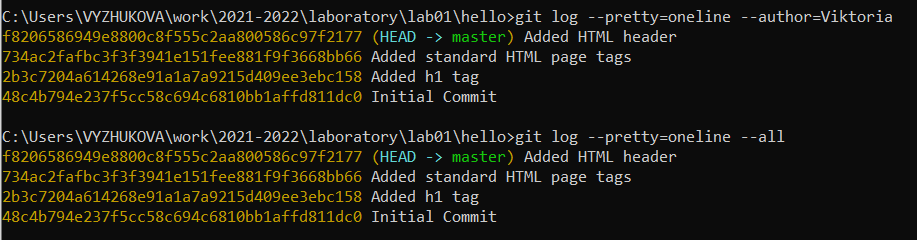
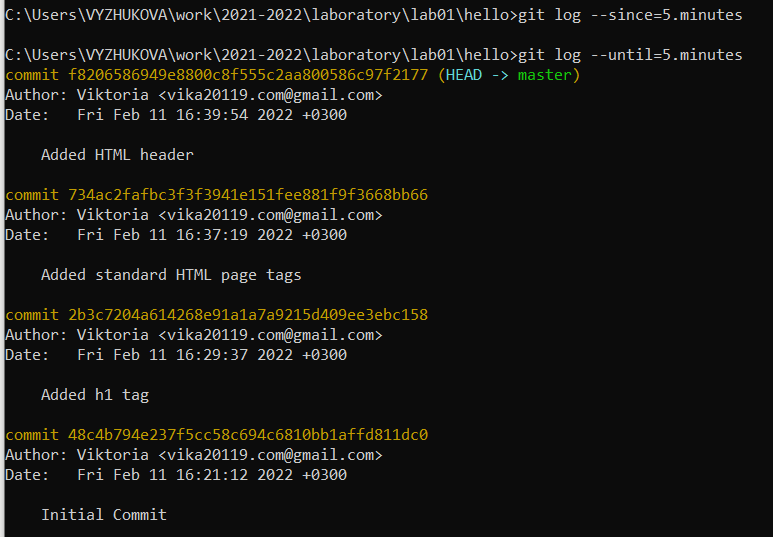


Делаю коммит второго изменения



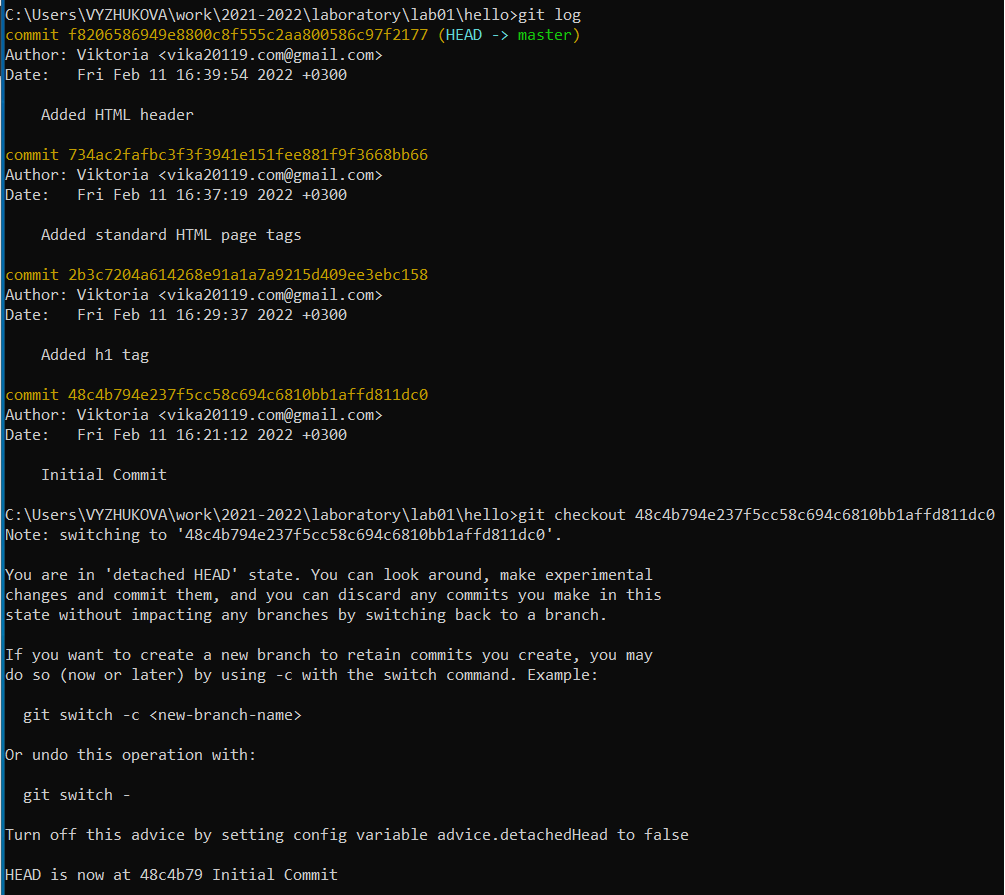
### 1.4.3 История

Получаю список произведенных изменений различными способами:

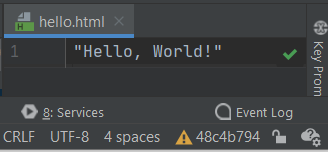
  
  
  


### 1.4.4 Получение старых версий

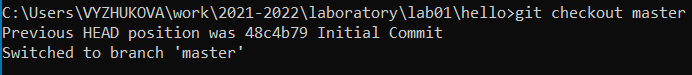
Получаю хэши предыдущих версий. Изучила данные лога, нашла хэш для первого коммита и переключилась на него.



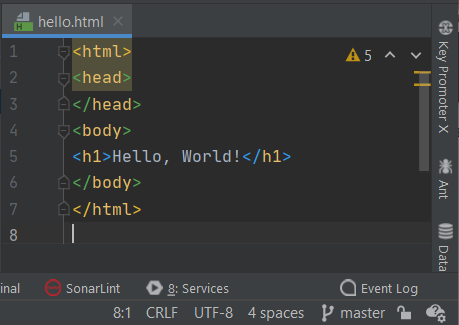
Посмотрела содержимое файла:



Вернулась к последней версии в ветке master



Содержимое файла в ветке мастер:

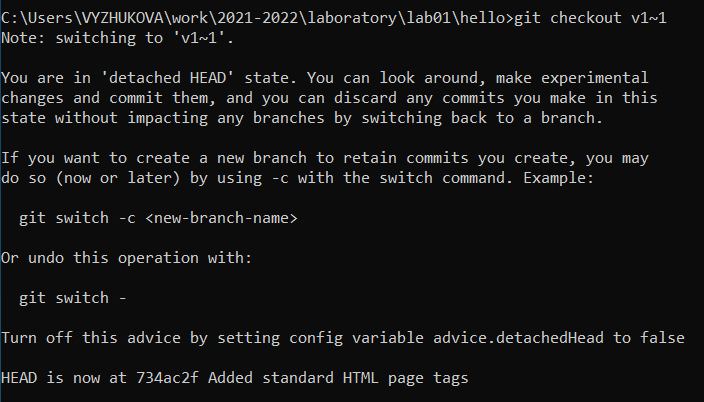


### 1.4.5 Создание тегов версий

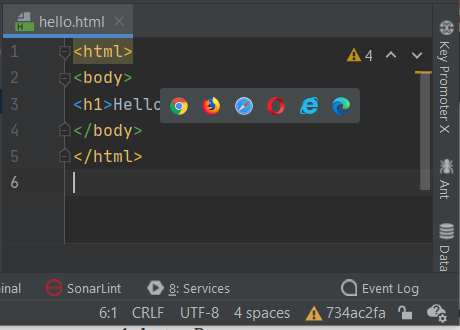
Давайте назовем текущую версию страницы hello первой (v1).  
Создаю тег первой версии



Теперь текущая версия страницы называется v1.  
Теги для предыдущих версий Давайте создадим тег для версии, которая идет  
перед текущей версией и назовем его v1-beta. В первую очередь переключаюсь на предыдущую версию.



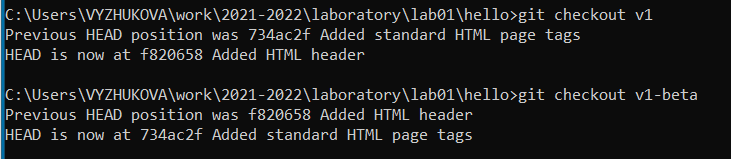
Содержимое файла в этой версии:



Это версия c тегами <html> и <body>, но еще пока без <head>. Сделаю  
ее версией v1-beta.

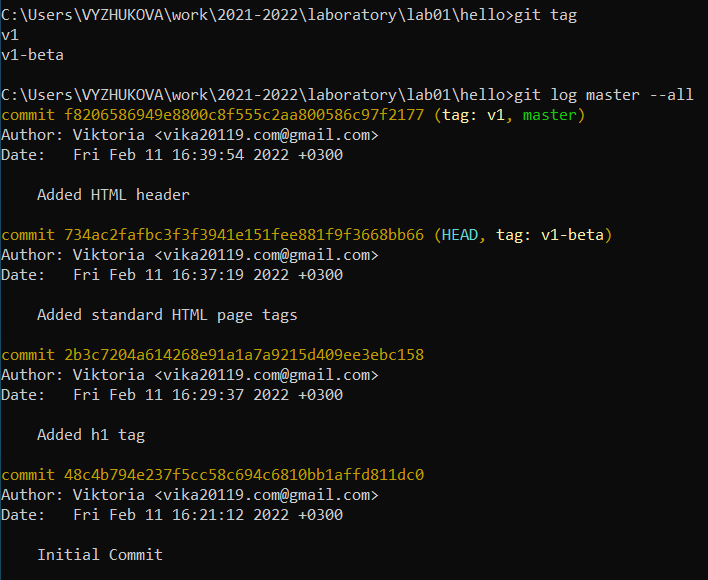


### 1.4.6 Переключение по имени тега

Теперь пробую попереключаться между двумя отмеченными версиями.  


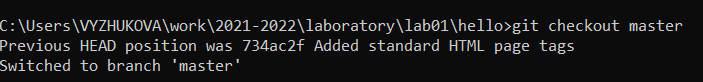
### 1.4.7 Просмотр тегов с помощью команды tag

Смотрю доступные теги и теги в логе

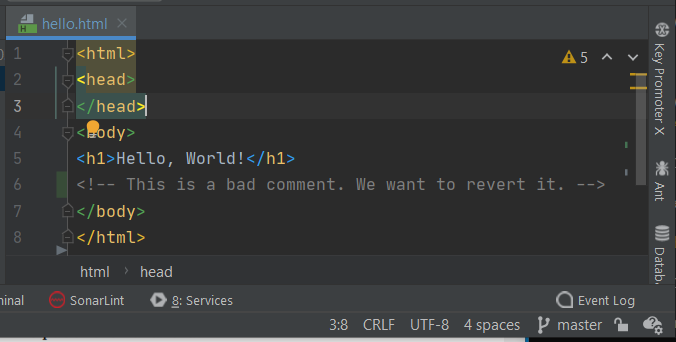


## 1.5 Отмена локальных изменений (до индексации)

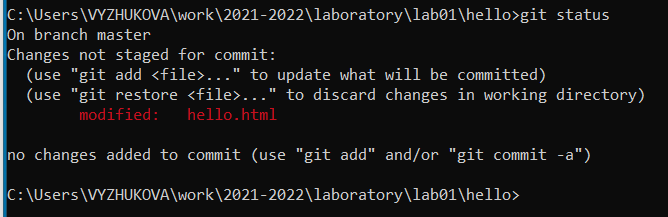
### 1.5.1 Переключение на ветку master

Переключаюсь на ветку мастер  


### 1.5.2 Измените hello.html

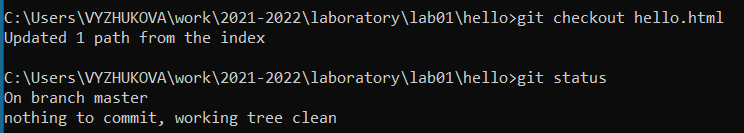
Вношу изменения в файл hello.html в виде нежелательного комментария.  


### 1.5.3 Проверьте состояние

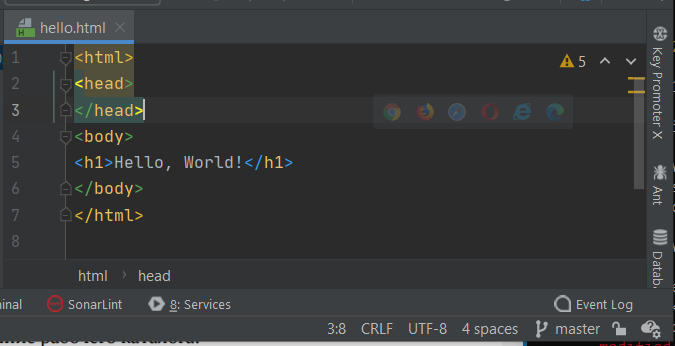
Сначала проверяю состояние рабочего каталога.  


Мы видим, что файл hello.html был изменен, но еще не проиндексирован.

### 1.5.4 Отмена изменений в рабочем каталоге

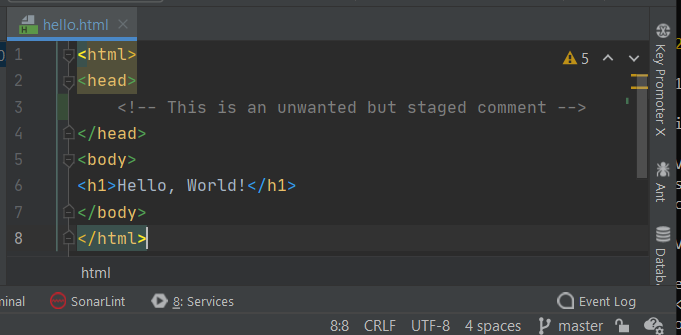
Использую команду git checkout для переключения версии файла  
hello.html в репозитории.  


Команда git status показывает нам, что не было произведено никаких изменений, не зафиксированных в рабочем каталоге.

Содержимое файла теперь:  


## 1.6 Отмена проиндексированных изменений (перед коммитом)

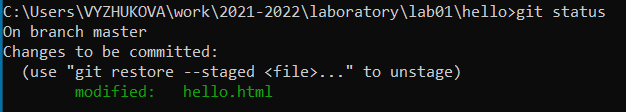
### 1.6.1 Изменение файла и индексирование изменений

Вношу изменение в файл hello.html в виде нежелательного комментария  


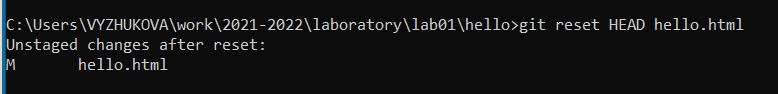
Индексирую это изменение.  
fig:

### 1.6.2 Проверка состояния

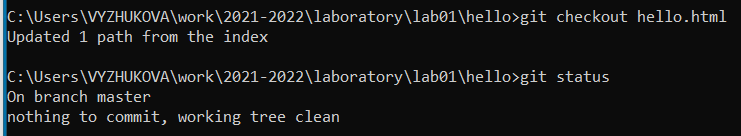
Проверяю состояние нежелательного изменения.

  
Состояние показывает, что изменение было проиндексировано и готово к коммиту.

### 1.6.3 Cброс буферной зоны

Выполняю сброс буферной зоны  


### 1.6.4 Переключение на версию коммита

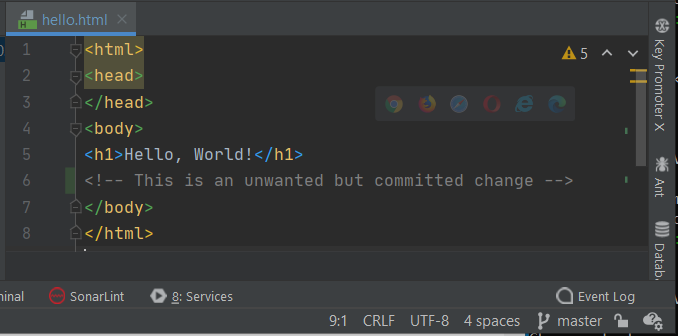
  
Наш рабочий каталог опять чист.

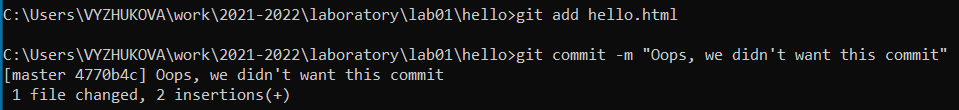
## 1.7 Отмена коммитов

### 1.7.1 Отмена коммитов

Мы отменим коммит путем создания нового коммита, отменяющего нежелательные изменения.

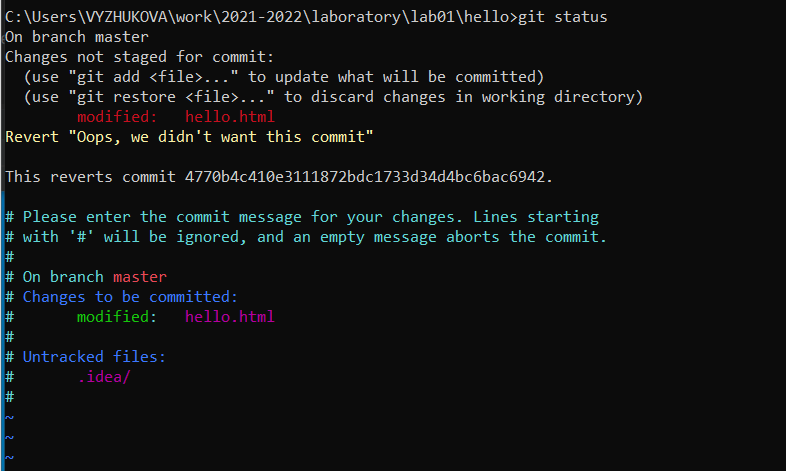
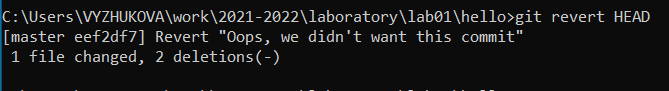
### 1.7.2 Измение файла и коммит

Изменяю файл hello.html на следующий.  


Выполняю индексацию и коммит  


### 1.7.3 Коммит с новыми изменениями, отменяющими предыдущие

Чтобы отменить коммит, нам необходимо сделать коммит, который удаляет изменения, сохраненные нежелательным коммитом.

Комментарий:  
  
Сам реверт:  


### 1.7.4 Проверка лога

Проверяю лог  


## 1.8 Удаление коммиттов из ветки

### 1.8.1 Команда git reset

При получении ссылки на коммит (т.е. хэш, ветка или имя тега), команда git  
reset:  
• перепишет текущую ветку, чтобы она указывала на нужный коммит;  
• опционально сбросит буферную зону для соответствия с указанным коммитом;  
• опционально сбросит рабочий каталог для соответствия с указанным коммитом.

### 1.8.2 Проверяю историю

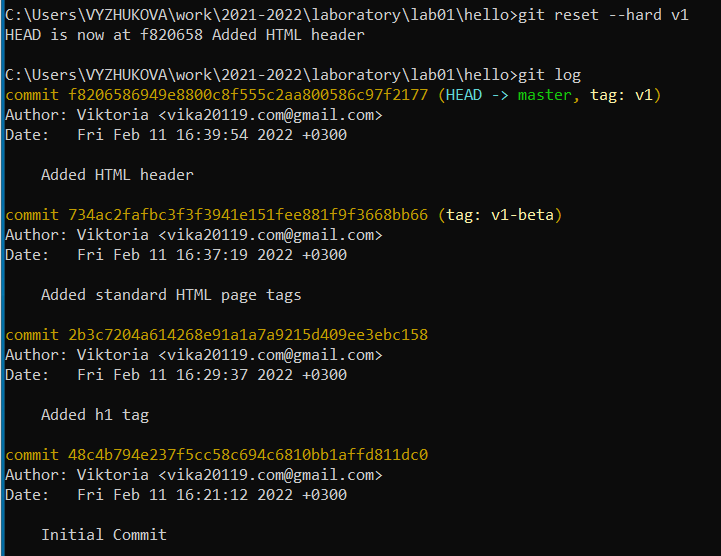
  
Мы видим, что два последних коммита в этой ветке — «Oops» и «Revert Oops».  
Удалим их с помощью сброса.

### 1.8.3 Для начала отметьте эту ветку

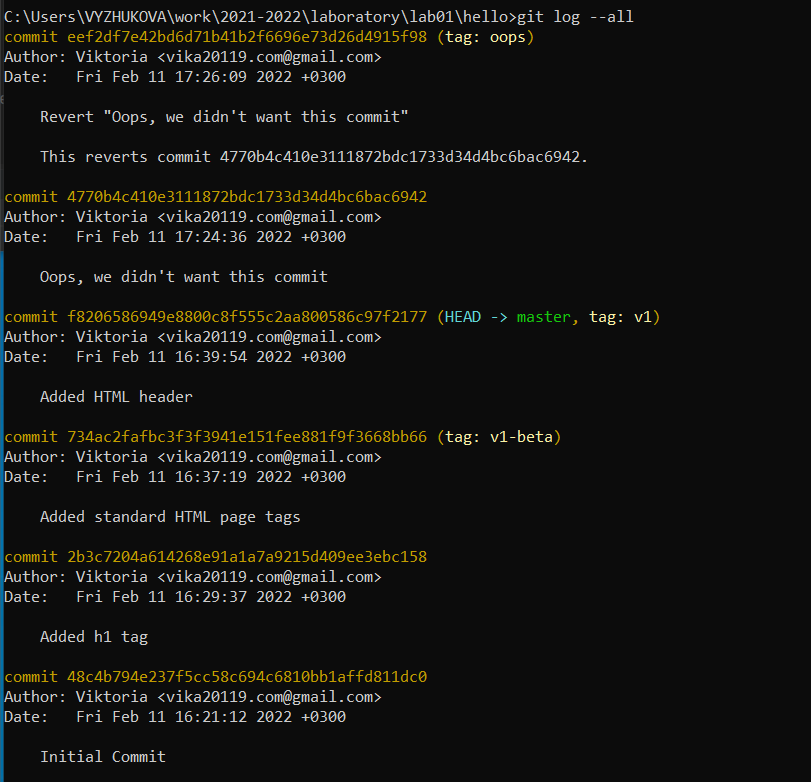
Но прежде чем удалить коммиты, отметим последний коммит тегом, чтобы  
потом можно было его найти.



### 1.8.4 Сброс коммитов к предшествующим коммиту Oops

Глядя на историю лога, мы видим, что коммит с тегом «v1» является коммитом,  
предшествующим ошибочному коммиту. Давайте сбросим ветку до этой точки.  
Наша ветка master теперь указывает на коммит v1, а коммитов Oops и Revert  
Oops в ветке уже нет.   


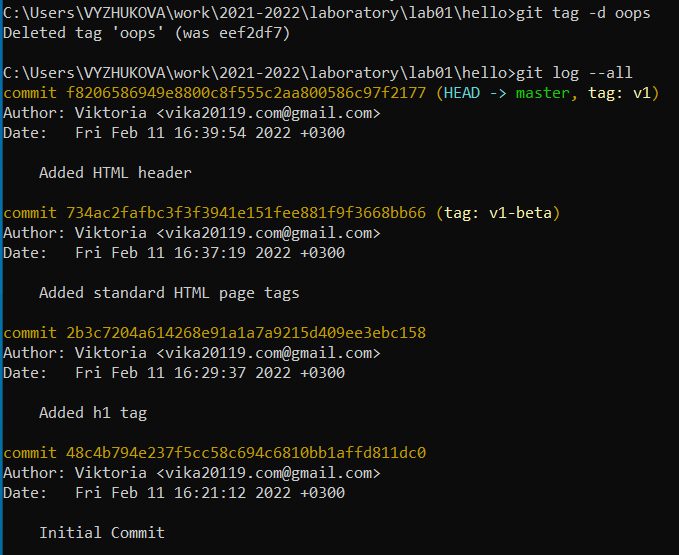
### 1.8.5 Ничего никогда не теряется

Все созданные коммиты можно посмотреть  


## 1.9 Удаление тега oops

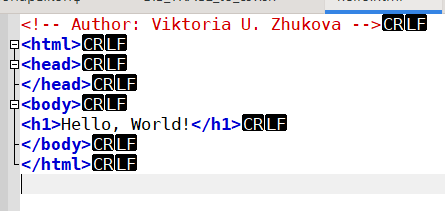
### 1.9.1 Удаление тега oops

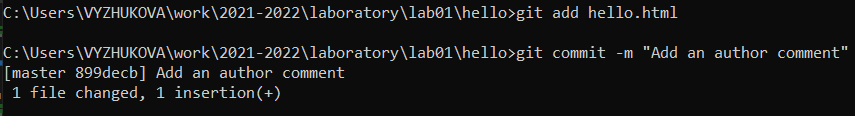
Тег oops свою функцию выполнил. Удалим его и коммиты, на которые он  
ссылался, сборщиком мусора.

  
Тег «oops» больше не будет отображаться в репозитории.

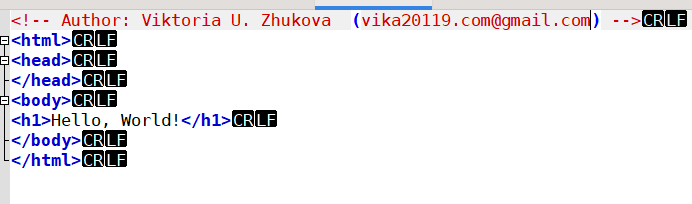
## 1.10 Внесение изменений в коммиты

### 1.10.1 Измение страницы, а затем коммит

Добавляю в страницу комментарий автора.  


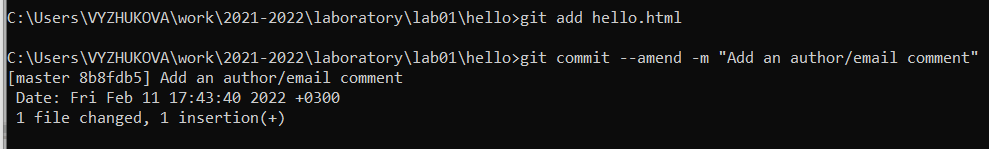
Индексирую и коммичу  


### 1.10.2 Необходим email

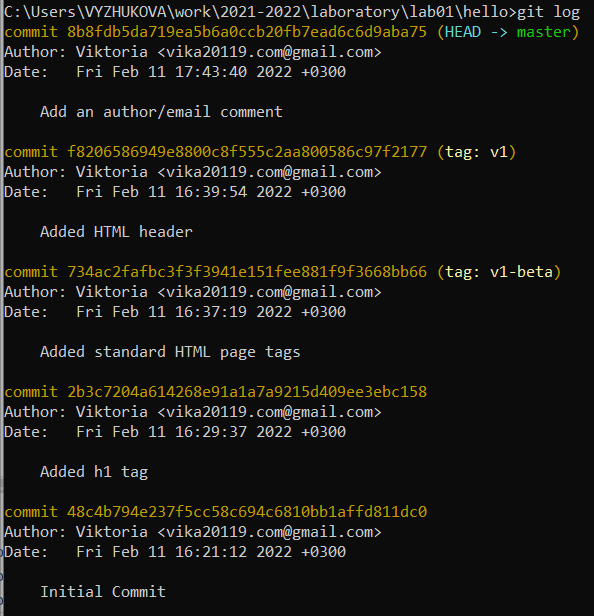
После совершения коммита понимаю, что любой хороший комментарий должен включать электронную почту автора. Обновляю страницу hello, включив в нее email.  


### 1.10.3 Измените предыдущий коммит

Не хочу создавать отдельный коммит только ради электронной почты. Изменю предыдущий коммит, включив в него адрес электронной  
почты.



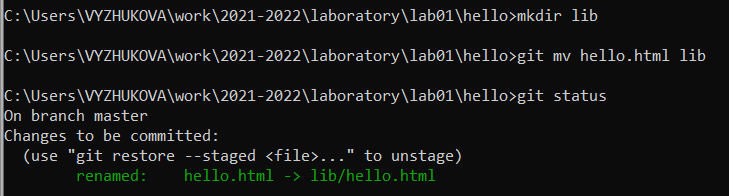
### 1.10.4 Просмотр истории



Мы можем увидеть, что оригинальный коммит «автор» заменен коммитом «автор/email».

## 1.11 Перемещение файлов

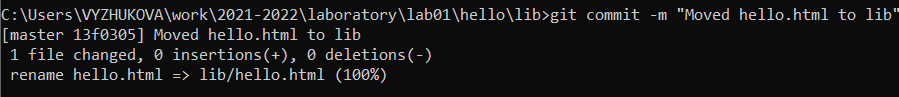
### 1.11.1 Переместите файл hello.html в каталог lib

Сейчас мы собираемся создать структуру нашего репозитория. Давайте перенесем  
страницу в каталог lib.  


## 1.12 Второй способ перемещения файлов

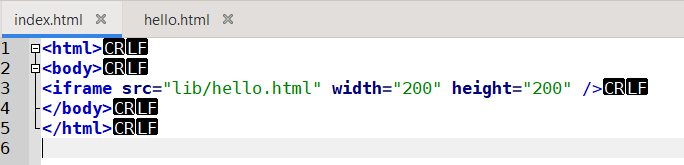
Следующий набор команд идентичен нашим последним действиям. Работы  
здесь побольше, но результат тот же.  
Мы могли бы выполнить:  
mkdir lib  
mv hello.html lib  
git add lib/hello.html  
git rm hello.html

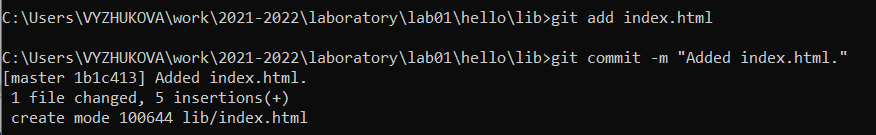
### 1.12.1 Коммит в новый каталог

Давайте сделаем коммит этого перемещения:  


## 1.13 Подробнее о структуре

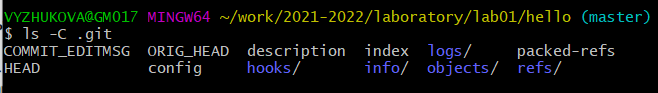
### 1.13.1 Добавление index.html

Добавим файл index.html в наш репозиторий  


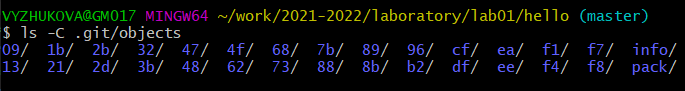
Добавим файл и сделаем коммит.  


### 1.14 Git внутри: Каталог .git

## 1.14.1 Каталог .git



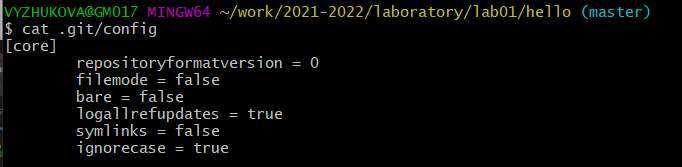
### 1.14.2 База данных объектов



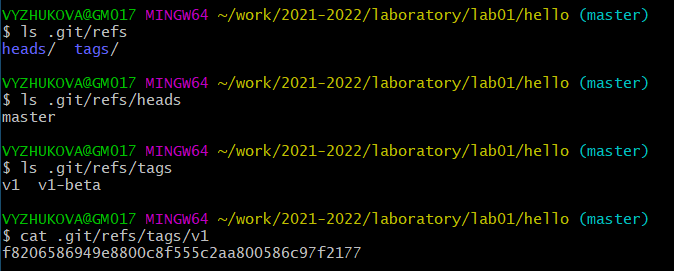
### 1.14.3 Углубляемся в базу данных объектов



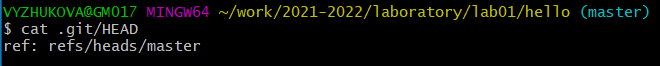
### 1.14.4 Config File



### 1.14.5 Ветки и теги

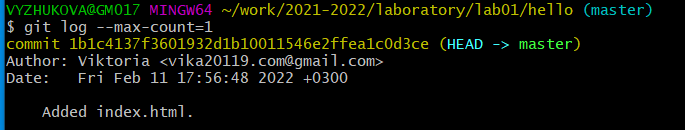


### 1.14.6 Файл HEAD

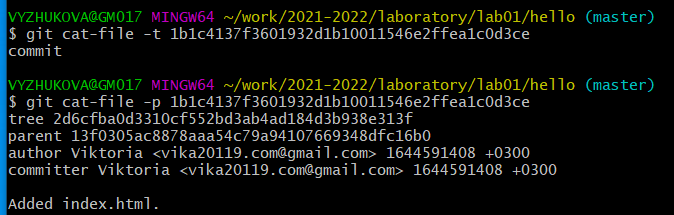


## 1.15 Работа непосредственно с объектами git

### 1.15.1 Поиск последнего коммита



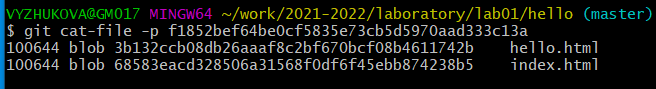
### 1.15.2 Вывод последнего коммита с помощью SHA1 хэша



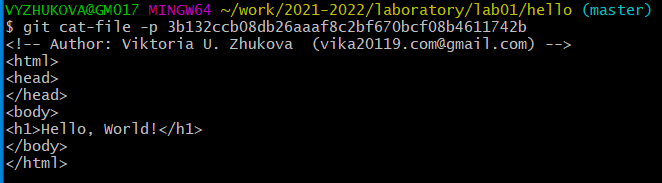
### 1.15.3 Поиск дерева

Мы можем вывести дерево каталогов, ссылка на который идет в коммите. Это должно быть описание файлов (верхнего уровня) в нашем проекте (для конкретного  
коммита).   
fig:

### 1.15.4 Вывод каталога lib



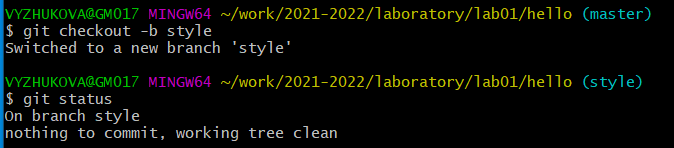
### 1.15.5 Вывод файла hello.html



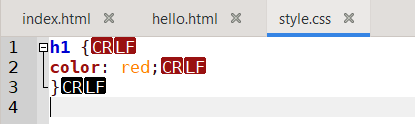
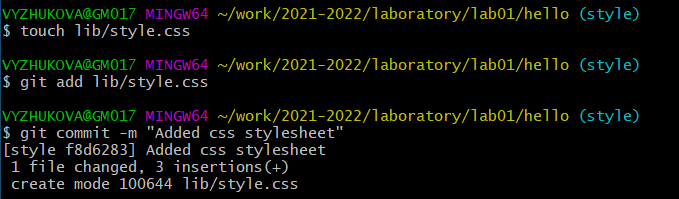
## 1.16 Создание ветки

Пора сделать наш hello world более выразительным. Так как это может занять  
некоторое время, лучше переместить эти изменения в отдельную ветку, чтобы  
изолировать их от изменений в ветке master.

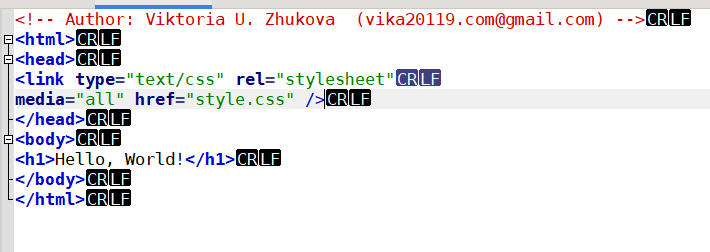
### 1.16.1 Создание ветки

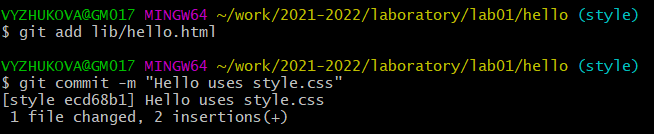
Давайте назовем нашу новую ветку «style».  


### 1.16.2 Добавьте файл стилей style.css

Добавляю новый файл стилей  
  
Индексирую и коммичу его  


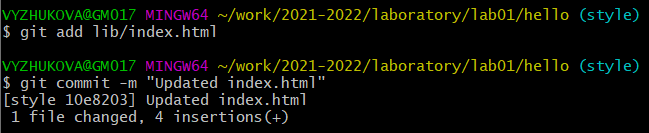
### 1.16.3 Измените основную страницу

Обновляю файл hello.html, чтобы использовать стили style.css.  


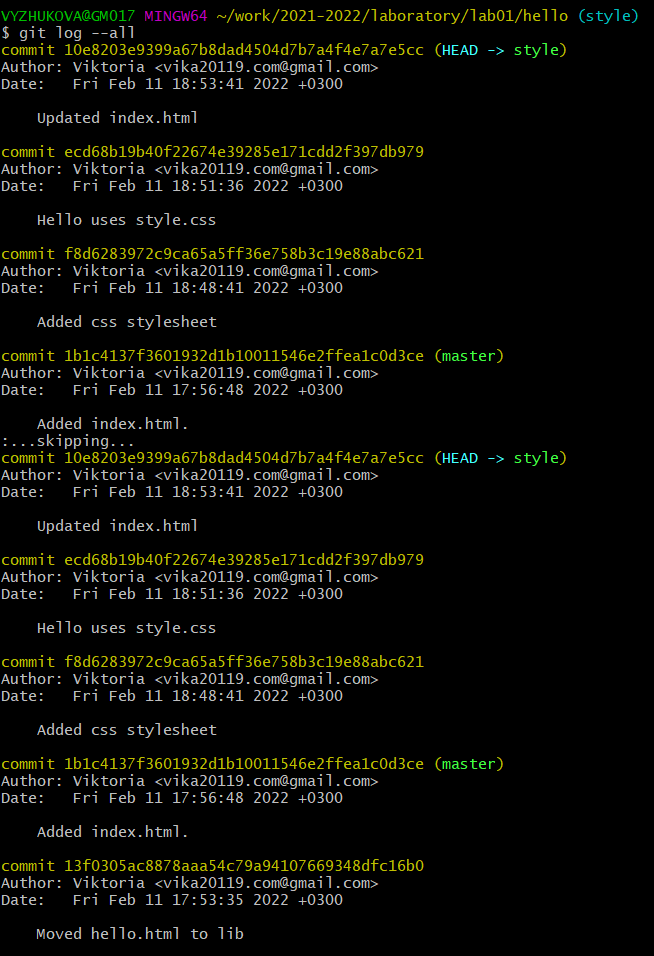
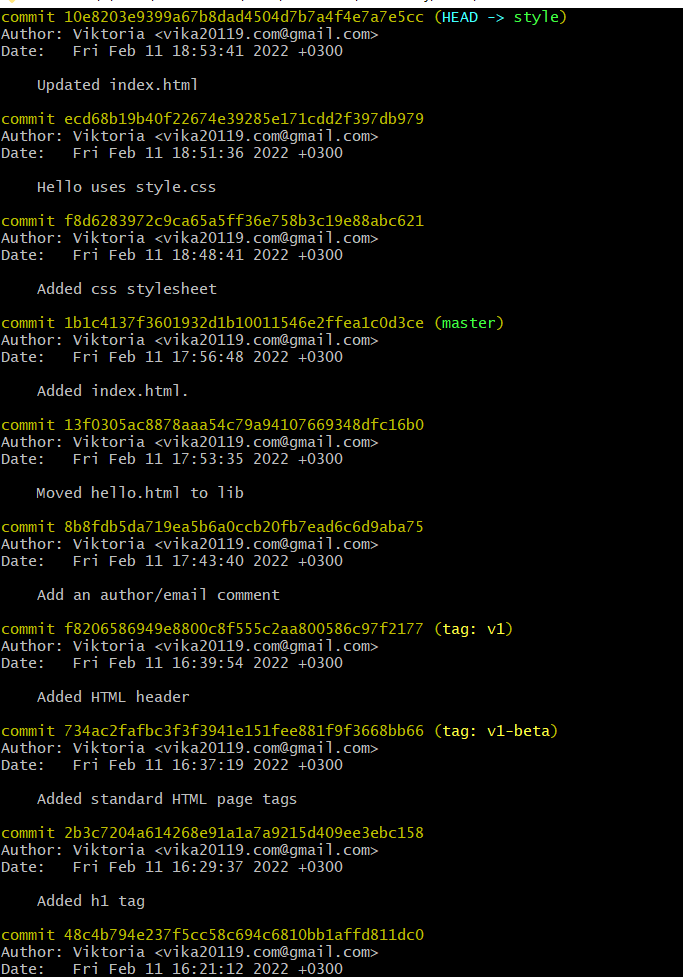
Индексирую и коммичу его  


### 1.16.4 Изменение index.html

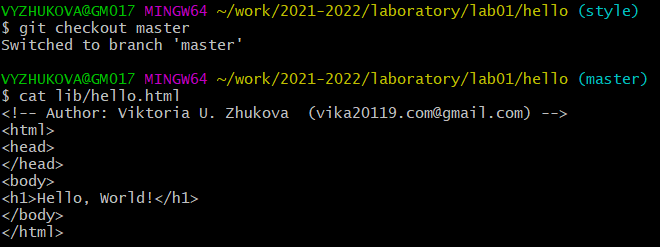
Обновляю файл index.html, чтобы он тоже использовал style.css  


Индексирую и коммичу его  


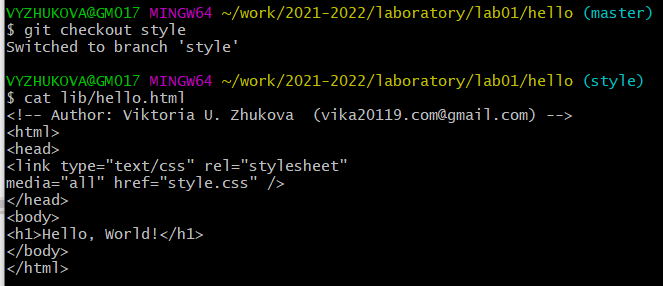
## 1.17 Навигация по веткам

Теперь в вашем проекте есть две ветки:  
  


### 1.17.1 Переключение на ветку master

Сейчас мы находимся на ветке master. Это заметно по тому, что файл  
hello.html не использует стили style.css.  


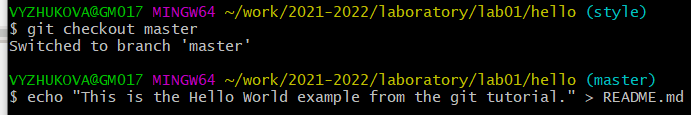
### 1.17.2 Вернемся к ветке style

Содержимое lib/hello.html подтверждает, что мы вернулись на ветку style.  


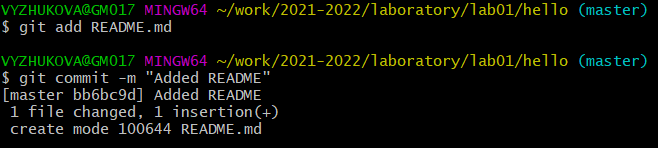
## 1.18 Изменения в ветке master

Пока вы меняли ветку style, кто-то решил обновить ветку master. Они добавили  
файл README.md.

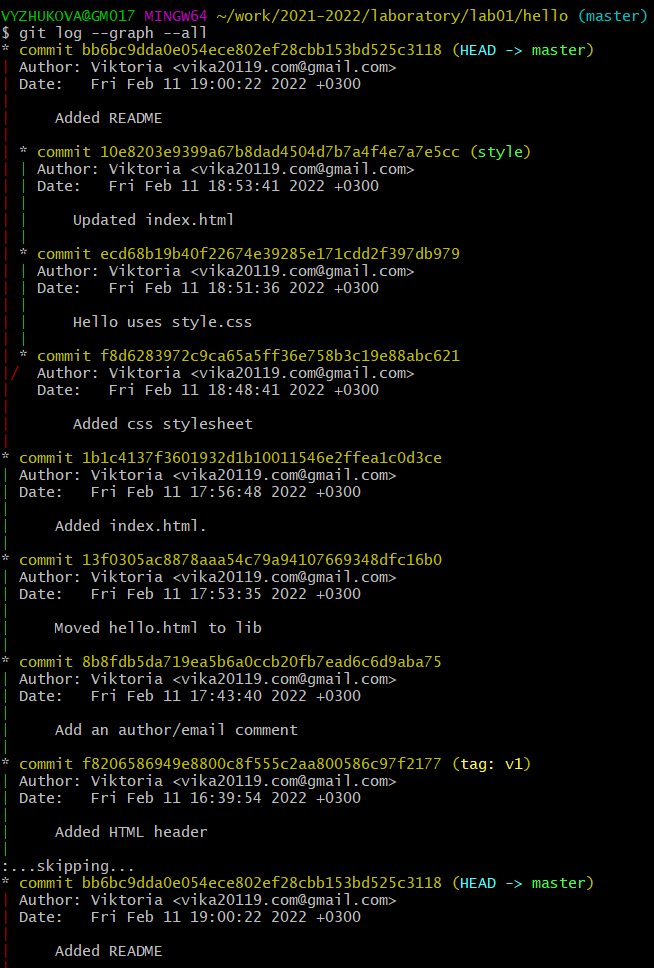
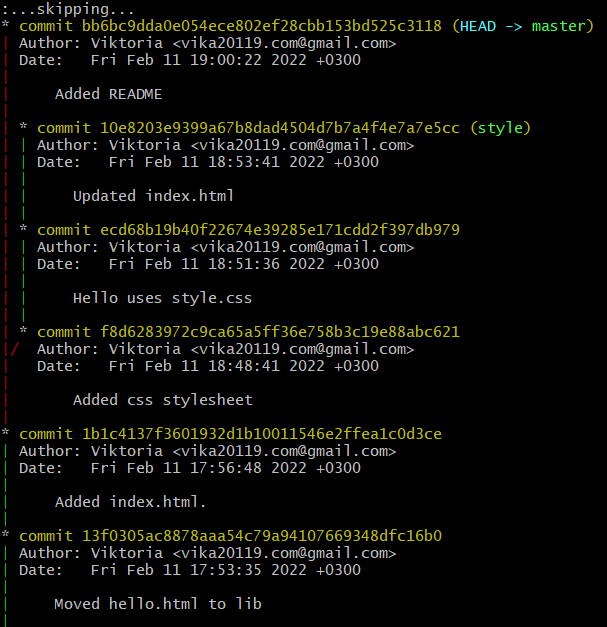
### 1.18.1 Создайте файл README в ветке master

Создаю файл README в ветке мастер  


## 1.19 Сделайте коммит изменений README.md в ветку master.

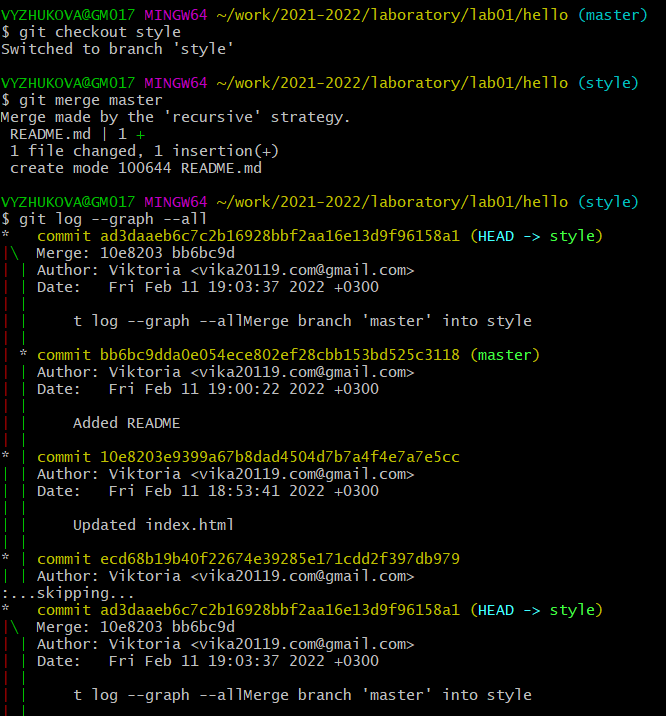


### 1.19.2 Просмотрите текущие ветки

Теперь у нас в репозитории есть две отличающиеся ветки.  
  


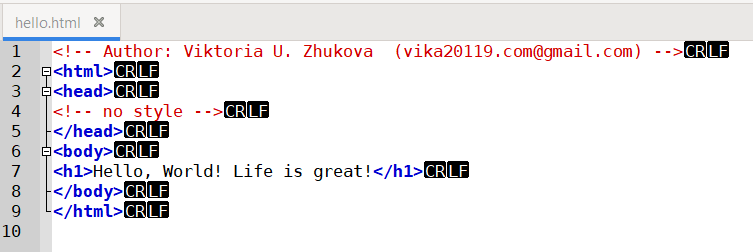
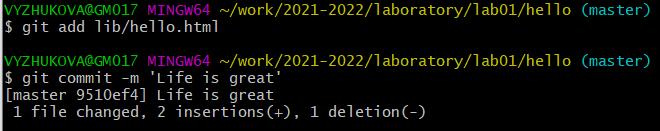
## 1.20 Слияние

### 1.20.1 Слияние веток

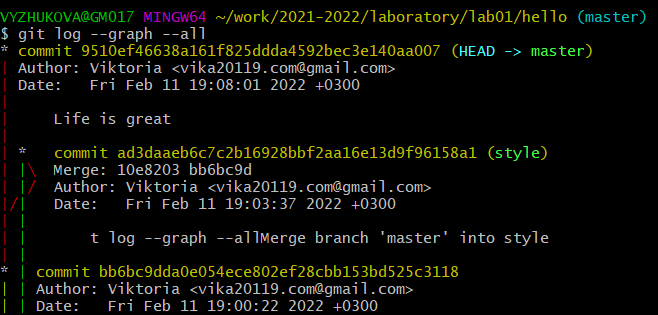
Слияние переносит изменения из двух веток в одну. Давайте вернемся к ветке  
style и сольем master с style.  


## 1.21 Создание конфликта

### 1.21.1 Вернитесь в master и создайте конфликт

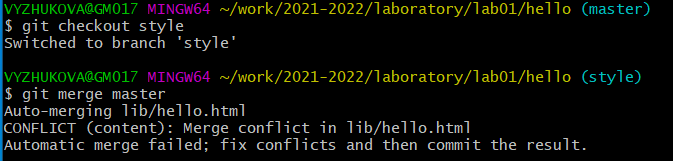
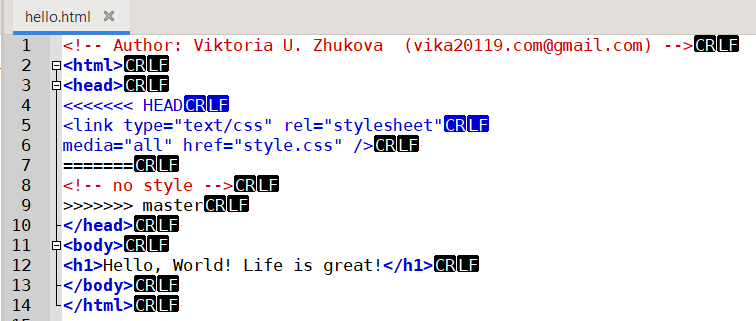
Вернусь в ветку master и внесу следующие изменения:  
  
Закоммичу их  


### 1.21.2 Просмотр веток

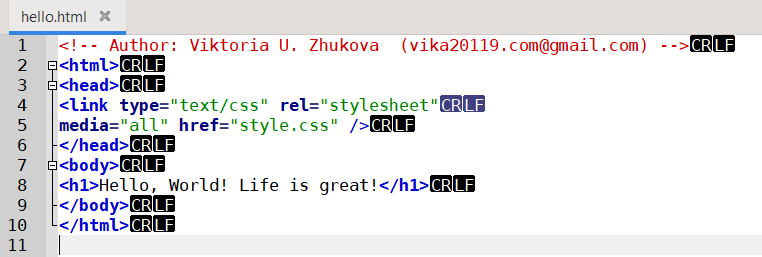


## 1.22 Разрешение конфликтов

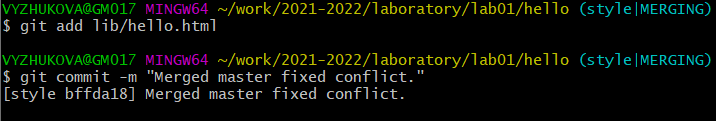
### 1.22.1 Слияние master с веткой style

Теперь вернемся к ветке style и попытаемся объединить ее с новой веткой  
master.  
  
Если открыть lib/hello.html, увидим:  


### 1.22.2 Решение конфликта



### 1.22.3 Сделайте коммит решения конфликта



### 1.22.4 Перебазирование как альтернатива слиянию

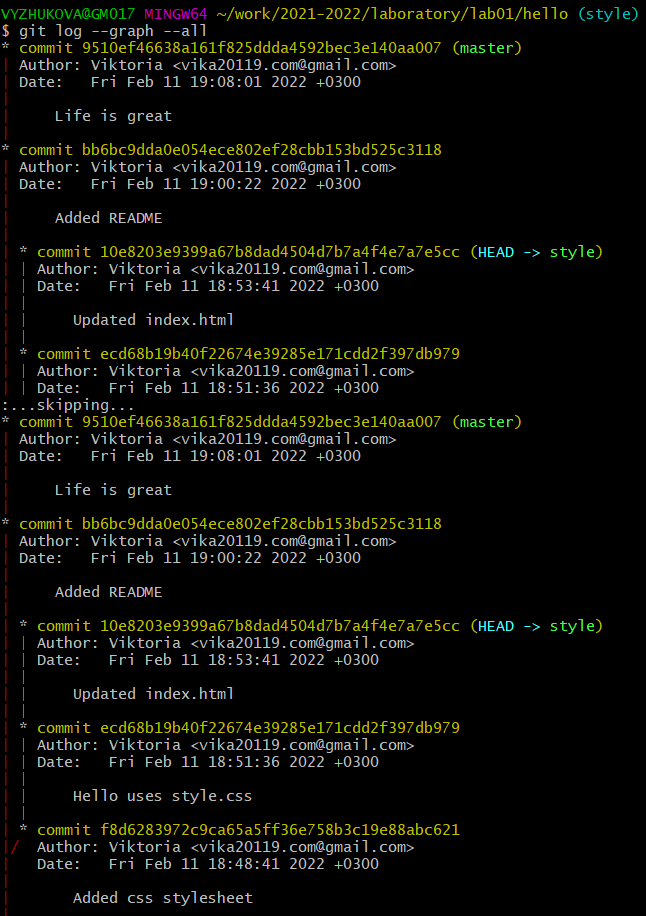
Рассмотрим различия между слиянием и перебазированием. Для того, чтобы это  
сделать, нам нужно вернуться в репозиторий в момент до первого слияния, а затем  
повторить те же действия, но с использованием перебазирования вместо слияния.  
Мы будем использовать команду reset для возврата веток к предыдущему  
состоянию.

## 1.23 Сброс ветки style

### 1.23.1 Сброс ветки style

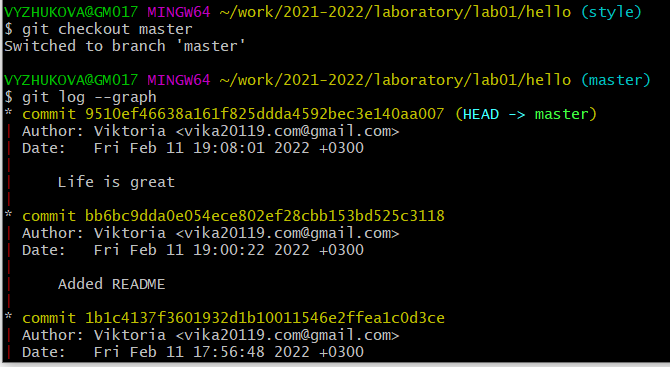
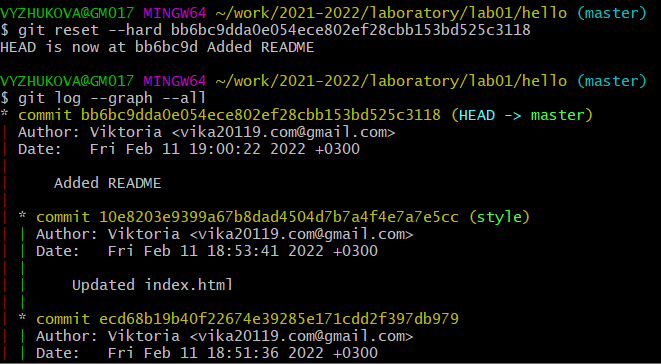


### 1.23.2 Проверьте ветку.

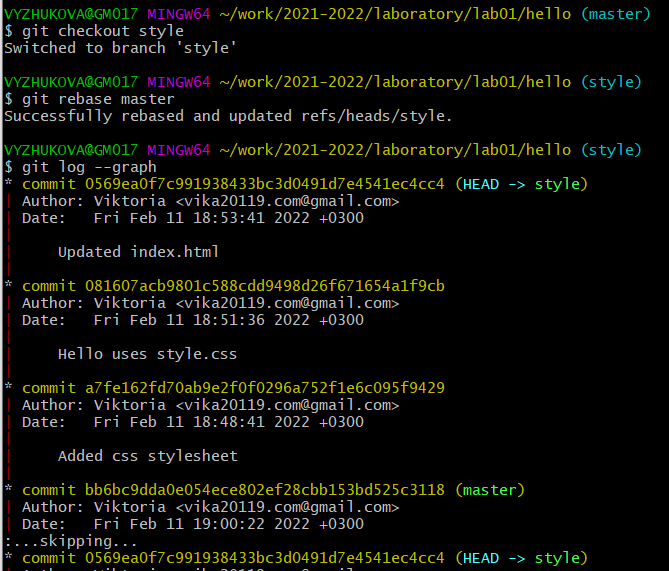
Лог ветки style. У нас в истории больше нет коммитов слияний.  


## 1.24 Сброс ветки master

### 1.24.1 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.  
  
Коммит «Added README» идет непосредственно перед коммитом конфликтующего интерактивного режима. Мы сбросим ветку master к коммиту «Added README».  


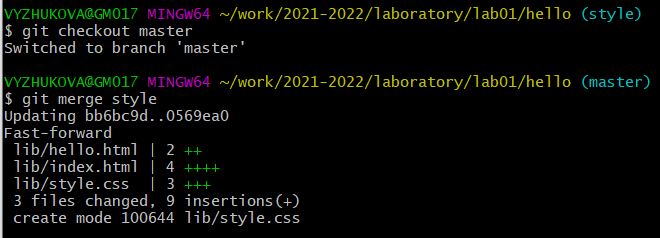
## 1.25 Перебазирование

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.  


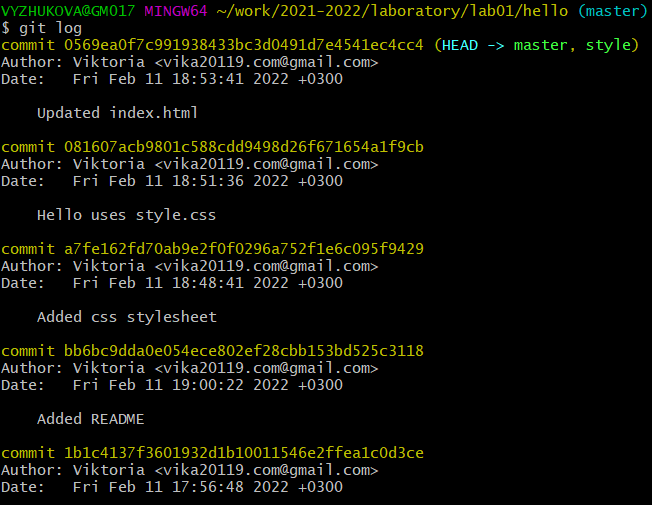
## 1.26 Слияние в ветку master

Мы поддерживали соответствие ветки style с веткой master (с помощью rebase),  
теперь давайте сольем изменения style в ветку master.

### 1.26.1 Слияние style в master

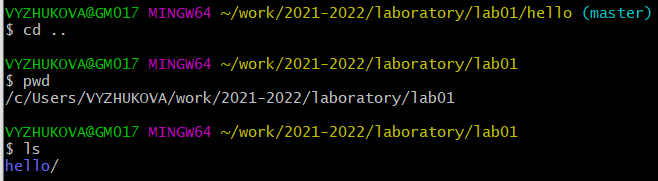


### 1.26.2 Просмотрите логи

  
Теперь ветки style и master идентичны.

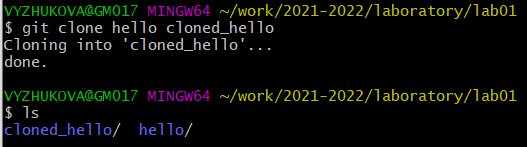
## 1.27 Клонирование репозиториев

### 1.27.1 Перейдите в рабочий каталог

Перехожу в рабочий каталог и делаю клон hello.  


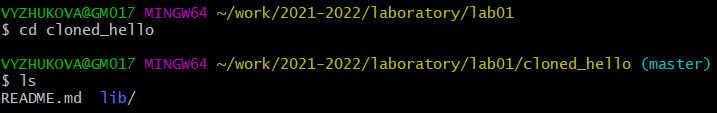
Сейчас мы находимся в рабочем каталоге.  
В этот момент вы должны находиться в «рабочем» каталоге. Здесь должен быть  
единственный репозиторий под названием «hello».

### 1.27.2 Создайте клон репозитория hello

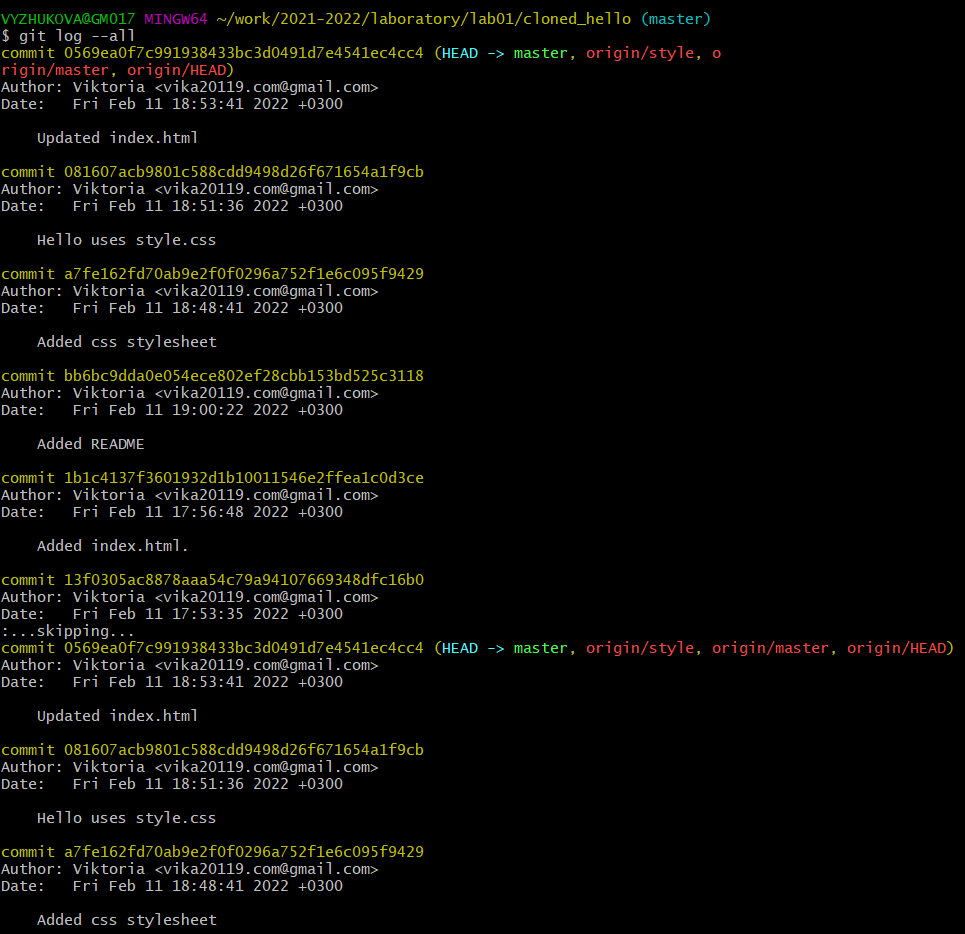
Создадим клон репозитория.  
  
В вашем рабочем каталоге теперь должно быть два репозитория: оригинальныйрепозиторий «hello» и клонированный репозиторий «cloned\_hello»

## 1.28 Просмотр клонированного репозитория

### 1.28.1 Давайте взглянем на клонированный репозиторий.



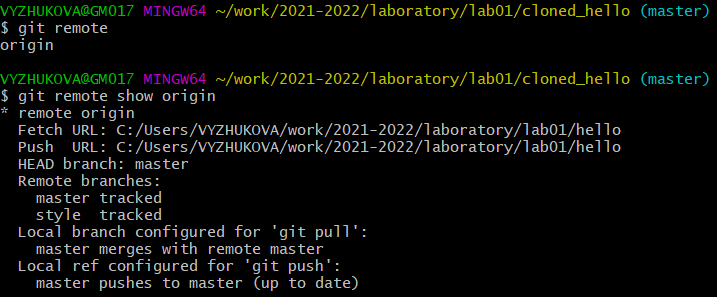
### 1.28.2 Просмотрите историю репозитория



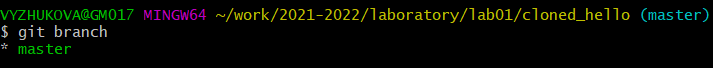
### 1.28.3 Удаленные ветки

Видим ветку master (HEAD) в списке истории. Также видим ветки со  
странными именами (origin/master, origin/style и origin/HEAD).

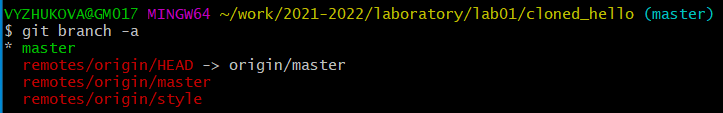
## 1.29 Что такое origin?

Мы видим, что клонированный репозиторий знает об имени по умолчанию  
удаленного репозитория. Давайте посмотрим, можем ли мы получить более подробную информацию об имени по умолчанию:  


## 1.30 Удаленные ветки

Давайте посмотрим на ветки, доступные в нашем клонированном репозитории.  
Как мы видим, в списке только ветка master. Где ветка style? Команда git  
branch выводит только список локальных веток по умолчанию.  


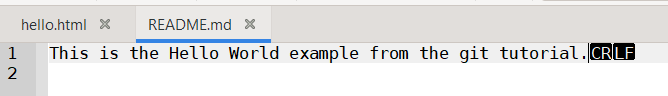
### 1.30.1 Список удаленных веток

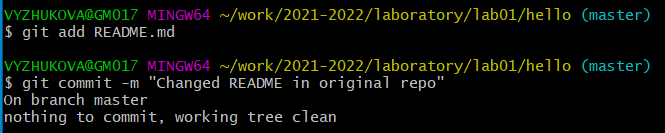
Все ветки:  


## 1.31 Изменение оригинального репозитория

Внесем некоторые изменения в оригинальный репозиторий, чтобы затем попытаться извлечь и слить изменения из удаленной ветки в текущую

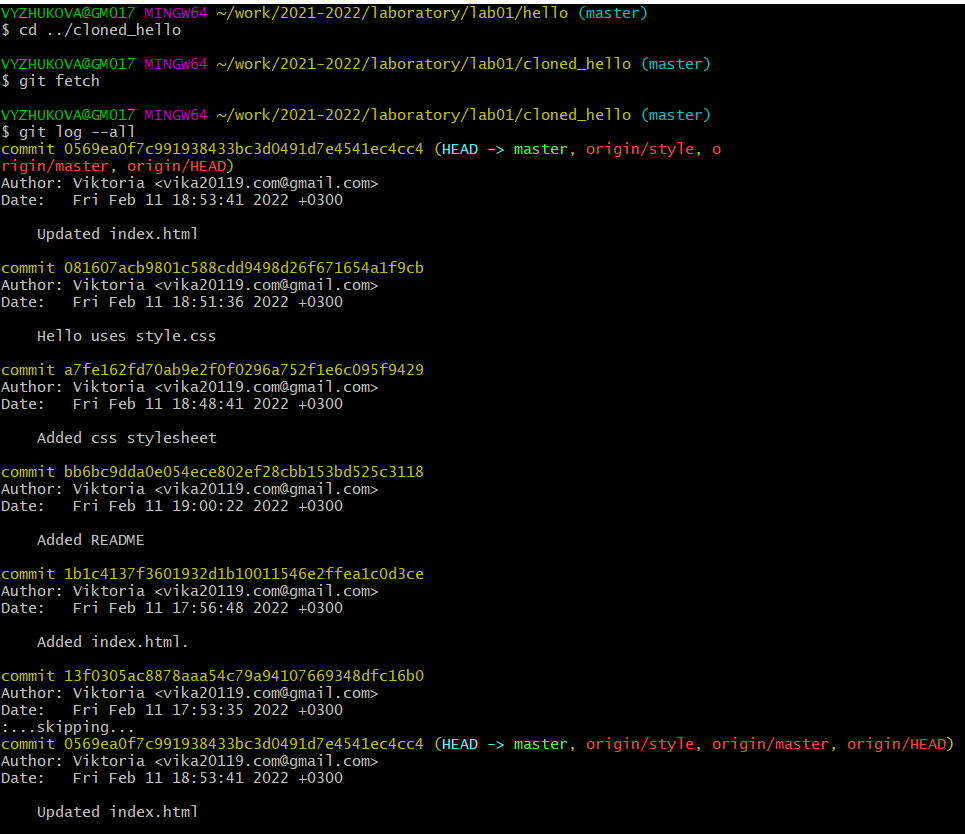
### 1.31.1 Внесем изменения в оригинальный репозиторий hello

Внесем следующие изменения в файл README:  


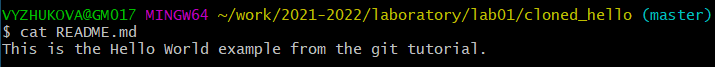
Теперь добавим это изменение и сделаем коммит  


Теперь в оригинальном репозитории есть более поздние изменения, которых  
нет в клонированной версии. Далее мы извлечем и сольем эти изменения в клонированный репозиторий.

### 1.31.2 Извлечение изменений

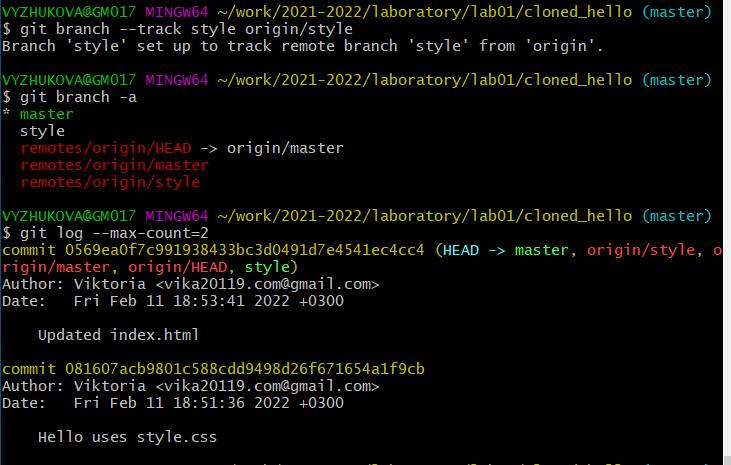
Научиться извлекать изменения из удаленного репозитория.  


### 1.31.3 Проверьте README.md

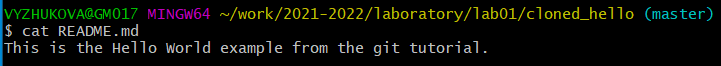


## 1.32 Слияние извлеченных изменений

### 1.32.1 Слейте извлеченные изменения в локальную ветку master



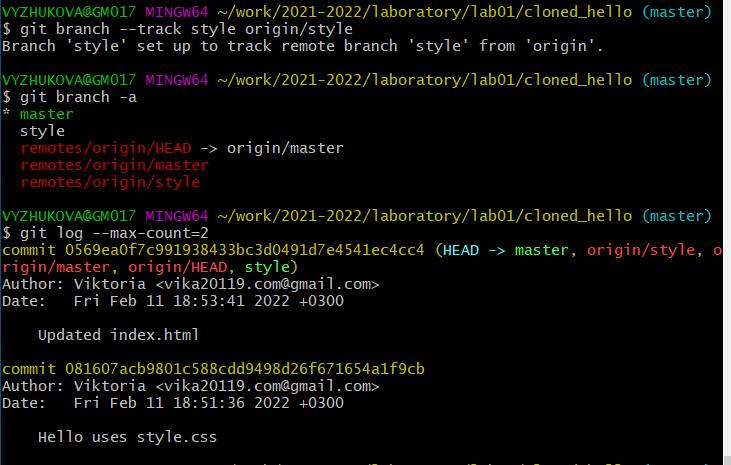
### 1.32.2 Еще раз проверьте файл README.md

Сейчас мы должны увидеть изменения.  


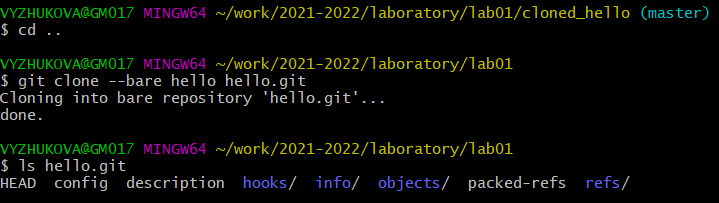
Теперь давайте рассмотрим объединение fetch и merge в одну команду. Выполнение:  
fig:

## 1.33 Добавление ветки наблюдения

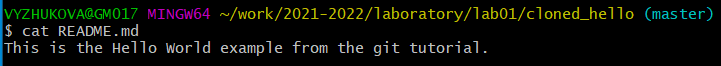
### 1.33.1 Добавьте локальную ветку, которая отслеживает удаленную ветку

  
Теперь мы можем видеть ветку style в списке веток и логе.

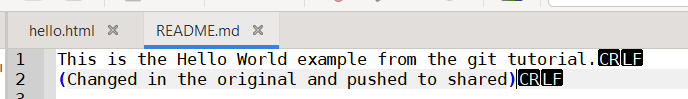
## 1.35 Создайте чистый репозиторий

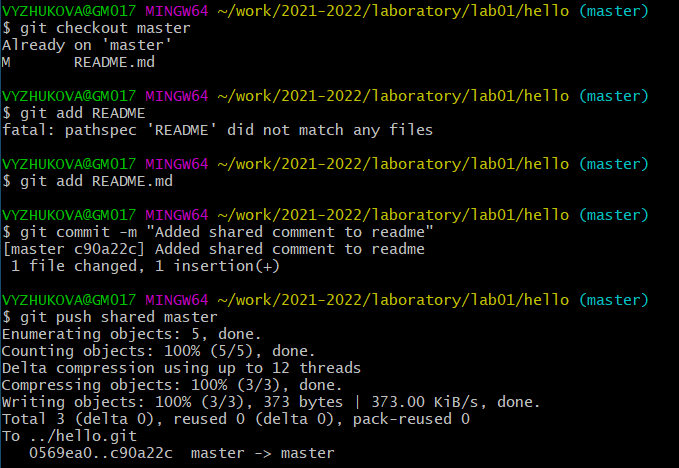
  
Сейчас мы находимся в рабочем каталоге  
Как правило, репозитории, оканчивающиеся на .git являются чистыми репозиториями. Мы видим, что в репозитории hello.git нет рабочего каталога. По сути,  
это есть не что иное, как каталог .git нечистого репозитория.

## 1.36 Добавление удаленного репозитория

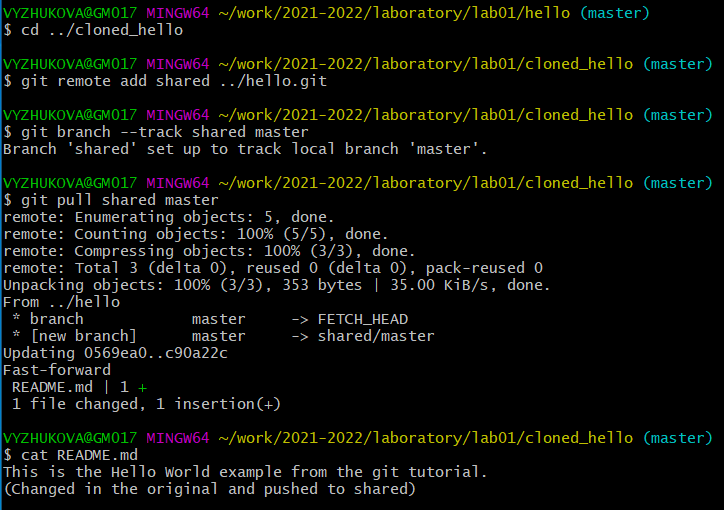
Давайте добавим репозиторий hello.git к нашему оригинальному репозиторию.  


## 1.37 Отправка 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. Отредактируйте файл README.md и  
сделайте коммит  
Файл README.md:  


Теперь отправим изменения в общий репозиторий.  


## 1.38 Извлечение общих изменений

Научимся извлекать изменения из общего репозитория. Переключусь в  
клонированный репозиторий и извлеку изменения, только что отправленные в  
общий репозиторий.  
Сейчас мы находимся в репозитории cloned\_hello.  


# Выводы

В результате данной работы я вспомнила основы git и научилась работать с файлами markdown.