\*\*МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ\*\*

**РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ**  
 **ВЫСШЕГООБРАЗОВАНИЯ«РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ»**  
 **Факультет физико-математических и естественных наук**

**Кафедра информационных технологий**

**ОТЧЕТ**  
 **по лабораторной работе 01**  
 **ТЕМА «Работа с git »**

**Выполнил/лa:**  
**Студент/ка группы:** НПИбд-02-21  
**Студенческий билет No:** 1032205421  
**Студент/кa:** Стелина Петрити

# Цель работы

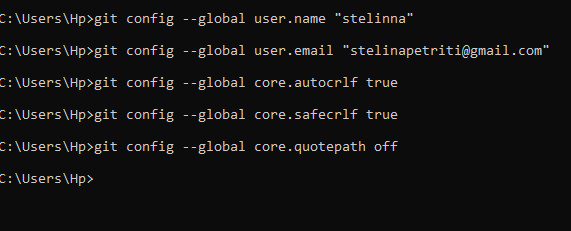
В этой лаборатории мы будем осваивать навыки работы с git

# Последовательность выполнения работы

## 1.1 Подготовка

### 1.1.1 Установка имени и электронной почты

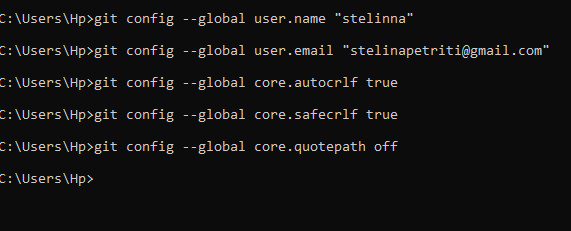
Если вы никогда ранее не использовали git, для начала вам необходимо осуществить установку. Выполните следующие команды, чтобы git узнал ваше имя и электронную почту. Если git уже установлен, можете переходить к разделу окончания строк.



*рис.1.1.1.1. имя и электронная почта*

### 1.1.2 Параметры установки окончаний строк

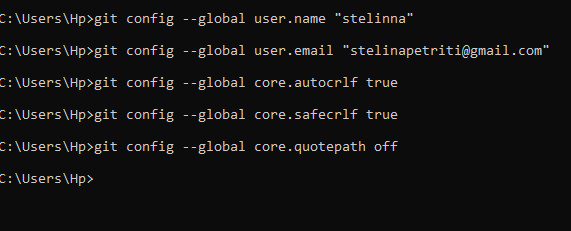
Для пользователей windows:



*рис.1.1.2.1. Настройка core.autocrlf с параметрами true и input все переводы строк текстовых файлов в главном репозитории одинаковы (git config --global core.autocrlf true,git config --global core.safecrlf true )*

### 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, установите соответствующий флаг.

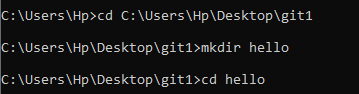


*рис.1.1.3.1.Настройка отображения в юникоде (git config --global core.quotepath off )*

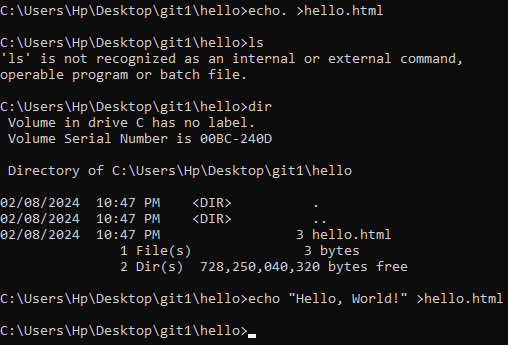
## 1.2 Создание проекта

### 1.2.1 Создайте страницу «Hello, World»

Начните работу в пустом рабочем каталоге с создания пустого каталога с именем hello, затем войдите в него и создайте там файл с именем hello.html.



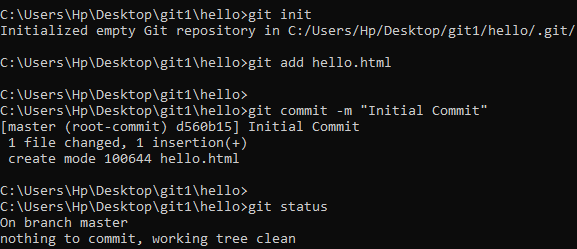
*рис.1.2.1.1.создание пустого каталога с именем hello(mkdir hello)*



*рис.1.2.1.2. создание файл с именем hello.html* *и написали "Hello, World!"с помощь echo*

### 1.2.2 Создание репозитория

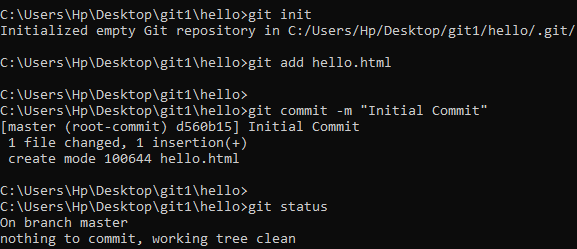
Чтобы создать git репозиторий из этого каталога, выполните команду git init.



*рис.1.2.2.1.создание git репозиторий из каталога hello\*\*(git init)*

### 1.2.3 Добавление файла в репозиторий

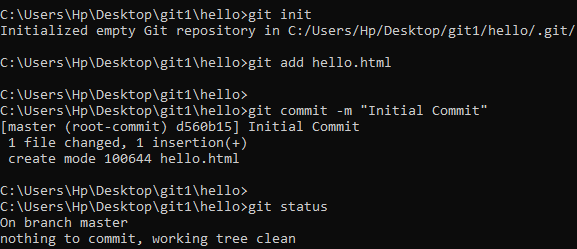
Добавим файл в репозиторий.  
git add hello.html  
git commit -m "Initial Commit"



*рис.1.2.3.1.Добавление файла в репозиторий (git add hello.html, git commit -m "Initial Commit" )*

### 1.2.4 Проверка состояние репозитория

Используйте команду git status, чтобы проверить текущее состояние репозитория.  
Команда проверки состояния сообщит, что комитет нечего. Это означает, что в репозитории хранится текущее состояние рабочего каталога, и нет никаких изменений, ожидающих записи.



*рис.1.2.4.1.проверка текущего состояния хранилища(git status)*

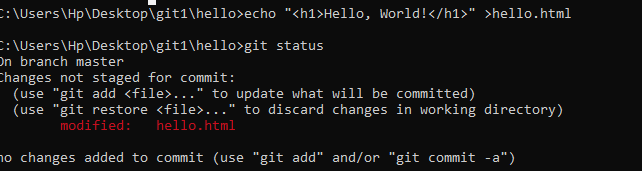
## 1.3 Внесение изменений

### 1.3.1 Измените страницу «Hello, World»

Добавим кое-какие HTML-теги к нашему приветствию. Измените содержимое  
файла hello.html на:

Hello, World!

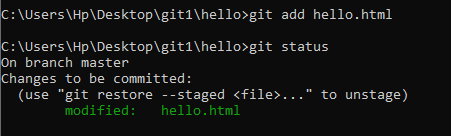
Проверьте состояние рабочего каталога.  
git знает, что файл hello.html был изменен, но при этом эти изменения еще не зафиксированы в репозитории.



*рис.1.2.4.1.Добавим HTML-теги (echo "Hello, World!">hello.html)*,*проверка текущего состояния (git status)*

### 1.4 Индексация изменений

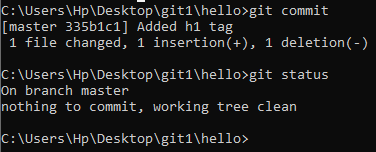
Теперь выполните команду git, чтобы проиндексировать изменения. Проверьте  
состояние.



*рис.1.4.1.выполните команду git, чтобы проиндексировать изменения(git add hello.html). Проверьте состояние(git status).*

#### 1.4.1 Коммит изменений

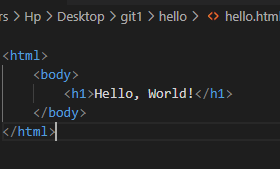
Сделайте коммит и проверьте состояние.



*рис.1.4.1.1. коммит(git commit) и проверка состояние(git status).*

#### 1.4.2 Добавьте стандартные теги страницы

Измените страницу «Hello, World», чтобы она содержала стандартные теги   
и .



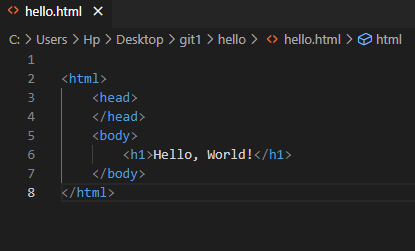
*рис.1.4.2.1.стандартные теги и .*

Теперь добавьте это изменение в индекс git

fig:

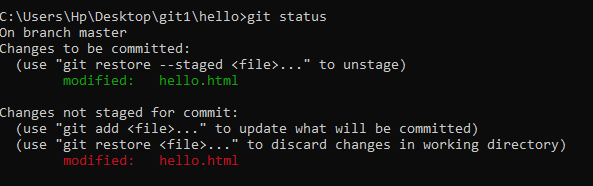
*рис.1.4.2.2.Добавим изменение в индекс git*

Теперь добавьте заголовки HTML (секцию ) к странице «Hello, World».



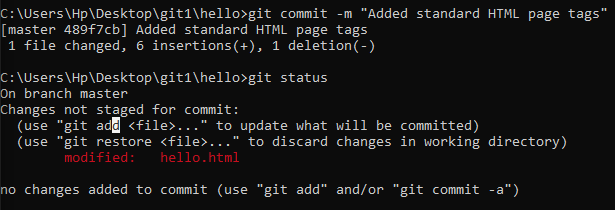
*рис.1.4.2.3. Добавим заголовки HTML.*

Проверьте текущий статус:



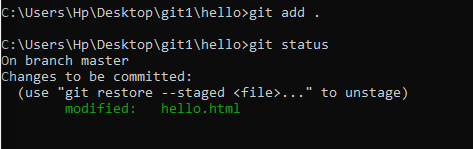
*рис.1.4.2.4.Проверьте текущий статус(git status)*

Произведите коммит проиндексированного изменения (значение по умолчанию), а затем еще раз проверьте состояние.



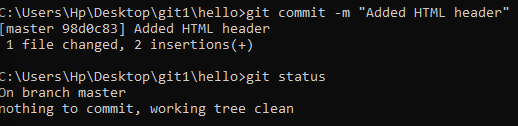
*рис.1.4.2.5.коммит(git commit -m "Added standard HTML page tags" ), проверьте состояние(git status).*

Теперь добавьте второе изменение в индекс, а затем проверьте состояние с помощью команды git status.



*рис.1.4.2.6. добавим второе изменение в индекс(git add .), проверьте состояние(git status*).

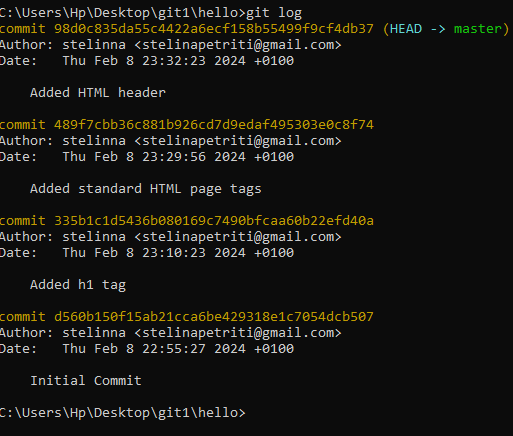
Второе изменение было проиндексировано и готово к коммиту. Сделайте коммит второго изменения



*рис.1.4.2.7.коммит второго изменения (git commit -m "Added HTML header")*

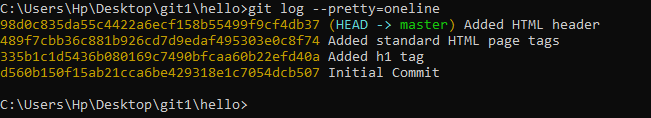
#### 1.4.3 История

Получим список произведенных изменений:  
git log



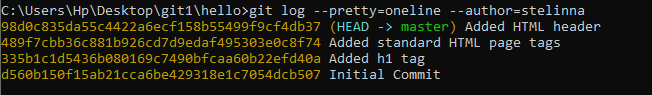
*рис.1.4.3.1.список произведенных изменений(git log)*

Однострочный формат истории:

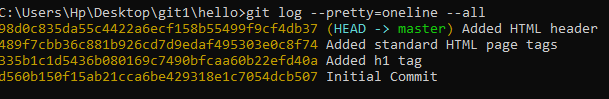


*рис.1.4.3.2. (git log --pretty=oneline)*

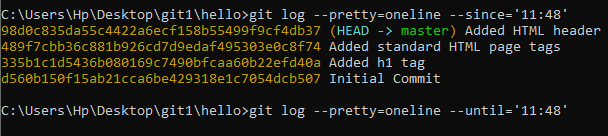
Вариантов отображения лога.



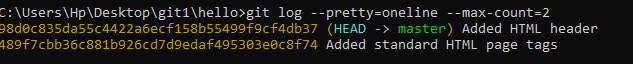
*рис.1.4.3.3. (git log --pretty=oneline --author= )*



*рис.1.4.3.4. (git log --pretty=oneline --all )*

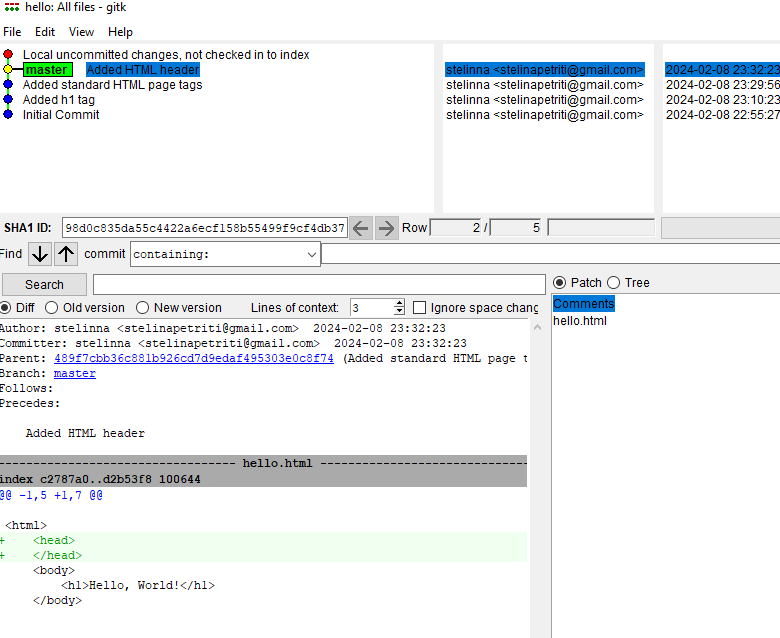


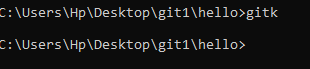
*рис.1.4.3.5. (git log --pretty=oneline --since='5 minutes ago', git log --pretty=oneline --until='5 minutes ago' )*



*рис.1.4.3.6.(git log --pretty=oneline --max-count=2 )*

Справочная страница:

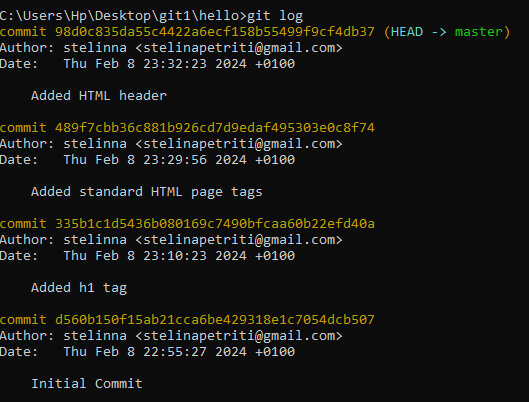




*рис.1.4.3.7. Инструмент gitk полезен в изучении истории изменений.*

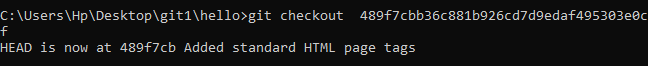
#### 1.4.4 Получение старых версий

Получите хэши предыдущих версий :

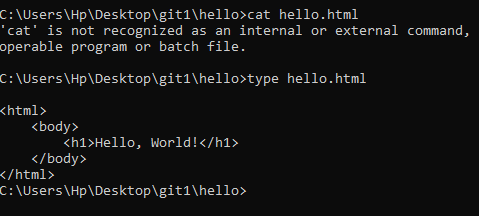


*рис.1.4.4.1.список произведенных изменений(git log)*

Используйте этот хэш-код (достаточно первых 7 знаков) в команде ниже. Затем проверьте содержимое файла hello.html

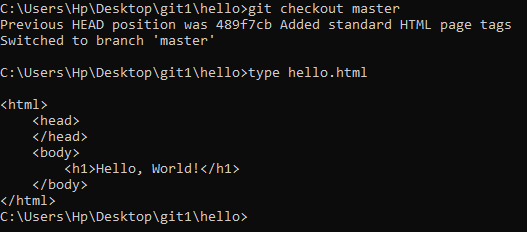


*рис.1.4.4.2.Используйте этот хэш-код(git check )*



*рис.1.4.4.3.проверьте содержимое файла hello.html (type hello.html)*

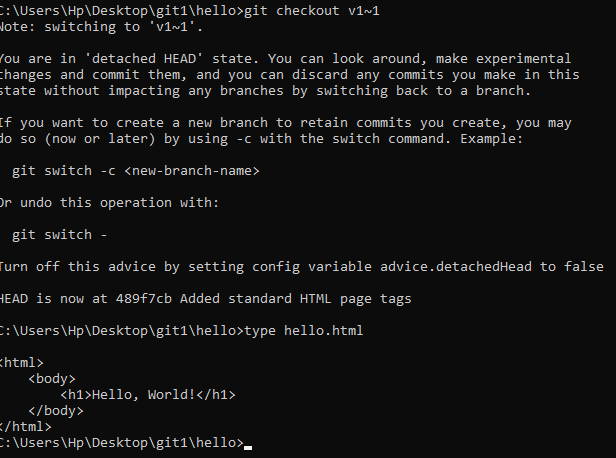
Вернитесь к последней версии в ветке master



*рис.1.4.4.4.Вернитесь к последней версии в master(git checkout master), проверьте содержимое файла(type hello.html)*

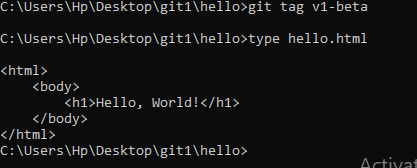
#### 1.4.5 Создание тегов версий

Давайте создадим тег для версии, которая идет перед текущей версией и назовем его v1-beta. В первую очередь нам надо переключиться на предыдущую версию. Вместо поиска до хэш, мы будем использовать ^, обозначающее «родитель v1». Вместо обозначения v1^ можно использовать v1~1. Это обозначение можно определить как «первую версию предшествующую  
v1».



*1.4.5.1. Создание тег для версии (git checkout v1~1), проверьте содержимое файла(type hello.html)*

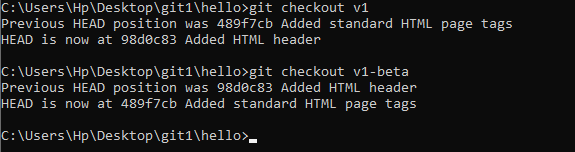
Это версия c тегами и , но еще пока без . Давайте сделаем  
ее версией v1-beta.



*1.4.5.2.Создание тег для версии(git tag v1-beta)*

#### 1.4.6 Переключение по имени тега

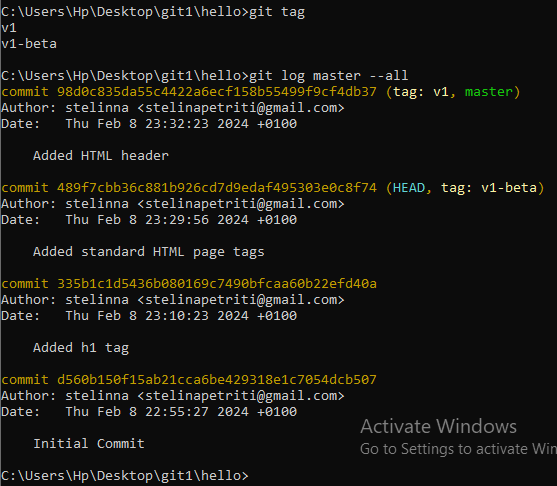
Теперь попробуйте попереключаться между двумя отмеченными версиями.



*\*1.4.6.1.попереключаться между двумя отмеченными версиями.*

#### 1.4.7 Просмотр тегов с помощью команды tag

Вы можете увидеть, какие теги доступны, используя команду git tag. Вы также можете посмотреть теги в логе.

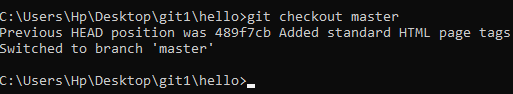


*1.4.6.1.увидим какие теги доступны(git tag), посмотреть теги в логе(git log master --all).*

### 1.5 Отмена локальных изменений (до индексации)

#### 1.5.1 Переключитесь на ветку master

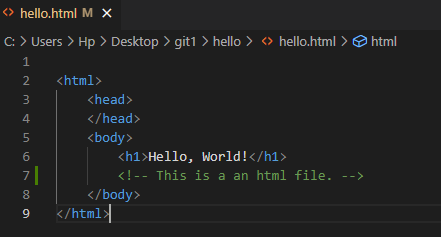
Убедитесь, что вы находитесь на последнем коммите ветки master, прежде чем  
продолжить работу.



*1.5.1.1.находитесь на последнем коммите ветки master(git checkout master)*

#### 1.5.2 Измените hello.html

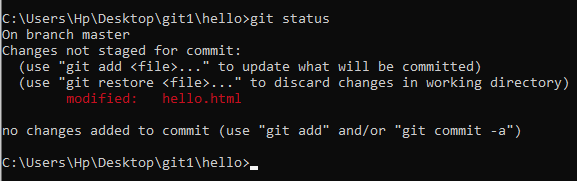
Внесите изменение в файл hello.html в виде нежелательного комментария.



*1.5.2.1. изменение в файл hello.html*

#### 1.5.3 Проверьте состояние

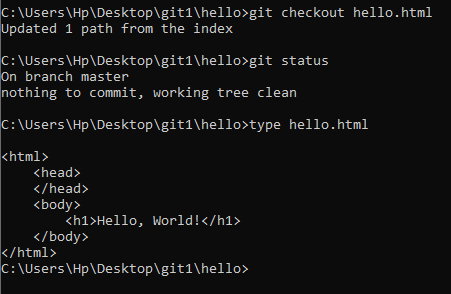
Сначала проверьте состояние рабочего каталога.



*1.5.3.1.проверьте состояние рабочего каталога.(git status)*

#### 1.5.4 Отмена изменений в рабочем каталоге

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



*1.5.4.1.командa git checkout hello.html для переключения файла в репозитории*

### 1.6 Отмена проиндексированных изменений (перед коммитом)

#### 1.6.1 Измените файл и проиндексируйте изменения

Внесите изменение в файл hello.html в виде нежелательного комментария



*1.6.1.1.комментария*

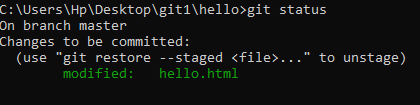
Проиндексируйте это изменение.

fig:

*1.6.1.1.(git add hello.html)*

#### 1.6.2 Проверьте состояние

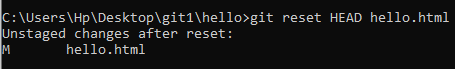
Проверьте состояние нежелательного изменения.



1.6.2.1.Проверьте состояние(git status)

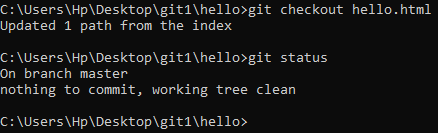
#### 1.6.3 Выполните сброс буферной зоны

К счастью, вывод состояния показывает нам именно то, что мы должны сделать  
для отмены индексации изменения.



*1.6.3.1.Выполнит сброс буферной зоны (git reset HEAD hello.html)*

#### 1.6.4 Переключитесь на версию коммита

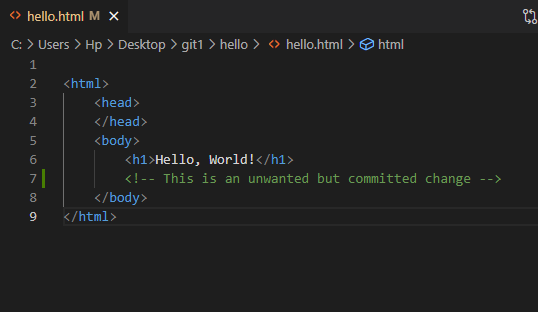


*1.6.4.1.Наш рабочий каталог опять чист(git checkout hello.html, git status).*

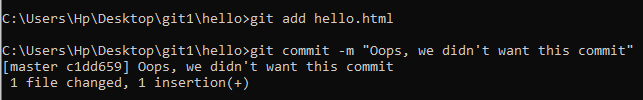
### 1.7 Отмена коммитов

#### 1.7.2 Измените файл и сделайте коммит

Измените файл hello.html на следующий.



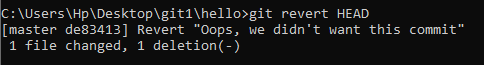
\*1.7.2.1.Измените файл hello.html



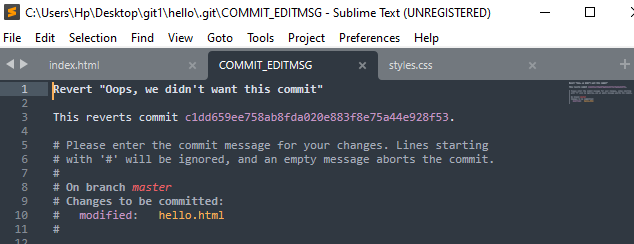
*1.7.2.2.Выполните: (git add hello.html, git commit -m "Oops, we didn't want this commit" )*

#### 1.7.3 Сделайте коммит с новыми изменениями, отменяющими предыдущие

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



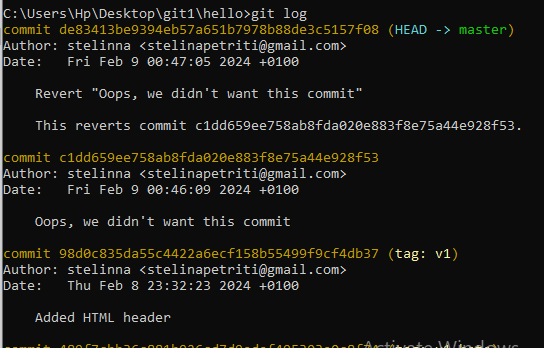
*1.7.3.1.удаляет изменения (git revert HEAD)*



1.7.3.2.Проверкa

#### 1.7.4 Проверьте лог

Проверка лога показывает нежелательные и отмененные коммиты в наш репозиторий.

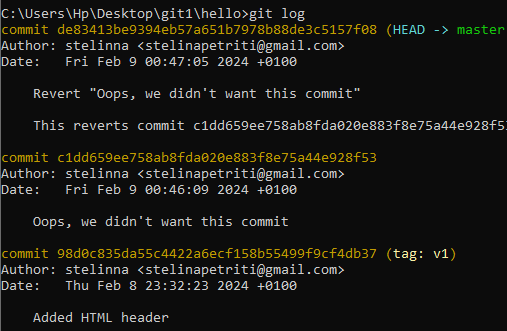


*1.7.4.1.Проверка лога (git log)*

### 1.8 Удаление коммиттов из ветки

#### 1.8.2 Проверьте нашу историю

Давайте сделаем быструю проверку нашей истории коммитов. Выполните:



*1.8.2.1.проверкa нашей истории коммитов(git log)*

#### 1.8.3. Для начала отметьте эту ветку

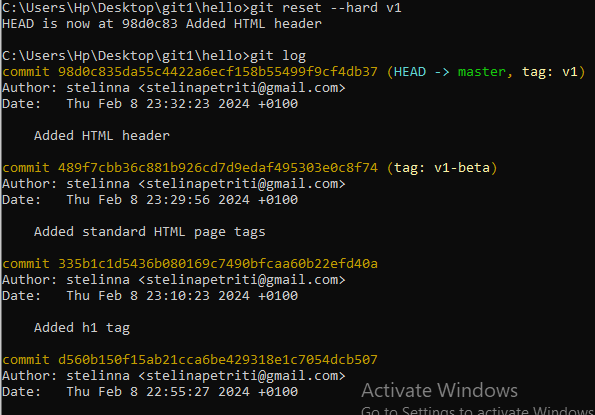
Но прежде чем удалить коммиты,давайте отметим последний коммит тегом,чтобы  
потом можно было его найти.

fig:

*1.8.3.1.отметим последний коммит тегом(git tag oops)*

#### 1.8.4 Сброс коммитов к предшествующим коммиту Oops

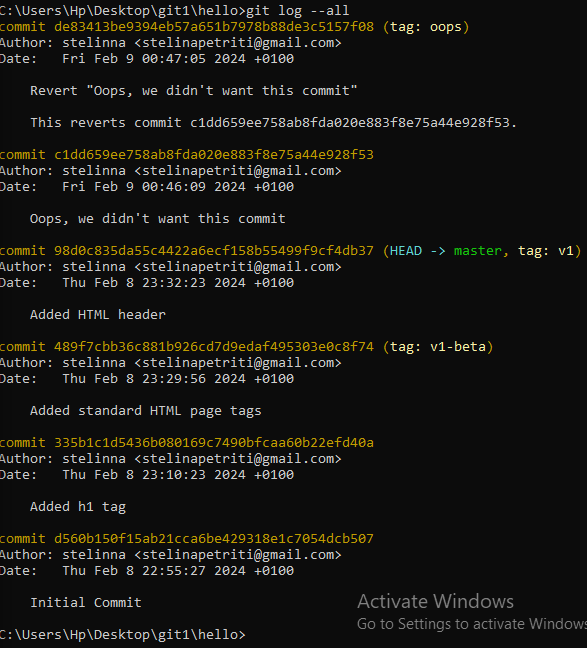
Давайте сбросим ветку до этой точки. Поскольку ветка имеет тег, мы можем использовать имя тега в команде сброса(если она не имеет тега, мы можем использовать хэш-значение).  
git reset --hard v1  
git log



*1.8.4.1.(git reset --hard v1git log)*

#### 1.8.5 Ничего никогда не теряется

Давайте посмотрим на все коммиты.

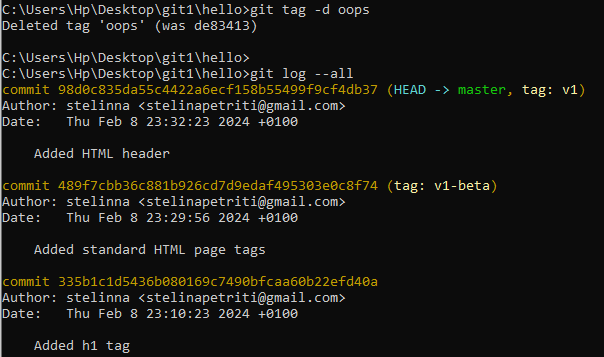


*1.8.5.1.посмотрим на все коммиты(git log --all )*

### 1.9 Удаление тега oops

#### 1.9.1 Удаление тега oops

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

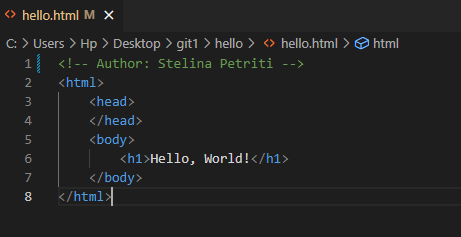


*1.9.1.1.удалим его и коммиты (git tag -d oops,git log --all )*

### 1.10 Внесение изменений в коммиты

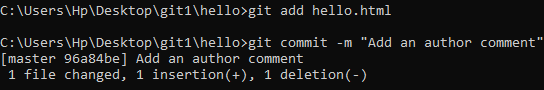
#### 1.10.1 Измените страницу, а затем сделайте коммит

Добавьте в страницу комментарий автора (вставьте свою фамилию).



*1.10.1.1. комментарии*

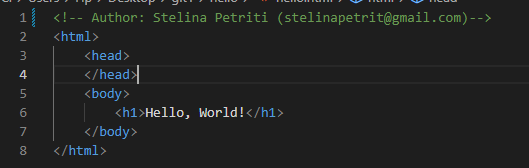
Выполните:



*1.10.1.2. (git add hello.html, git commit -m "Add an author comment")*

#### 1.10.2 Необходим email

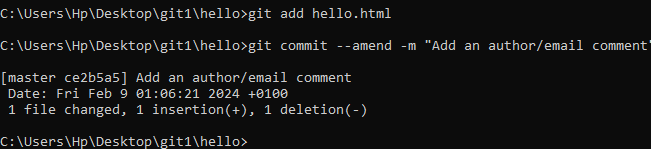
После совершения коммита вы понимаете, что любой хороший комментарий должен включать электронную почту автора. Обновите страницу hello, включив в нее email.



*1.10.2.1.oбновите страницу hello, включив в нее em*ail

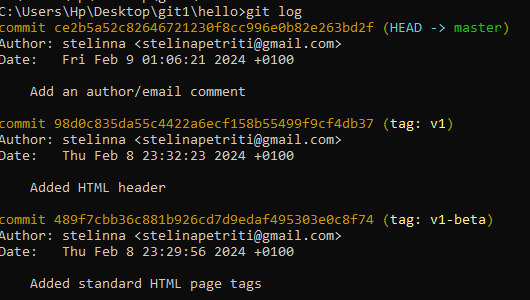
#### 1.10.3 Измените предыдущий коммит

Давайте изменим предыдущий коммит, включив в него адрес электронной  
почты.  
Выполните:



*1.10.3.1.(git add hello.html, git commit --amend -m "Add an author/email comment ")*

#### 1.10.4 Просмотр истории

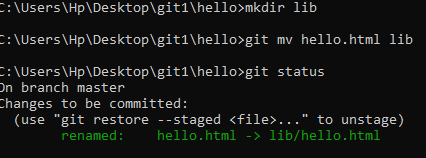


*1.10.4 Просмотр истории(git log)*

### 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.

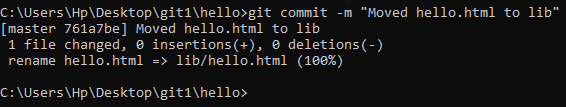


*1.11.1.1. создать структуру нашего репозитория, перенесем страницу в каталог lib. ( mkdir lib, git mv hello.html lib, git status)*

### 1.12 Второй способ перемещения файлов

#### 1.12.1 Коммит в новый каталог

Давайте сделаем коммит этого перемещения:

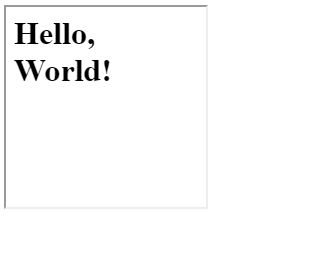


*1.12.1.1. сделаем коммит(git commit -m "Moved hello.html to lib" )*

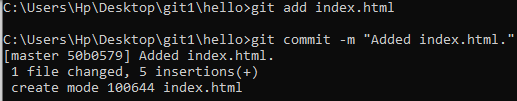
### 1.13 Подробнее о структуре

#### 1.13.1 Добавление index.html

Добавим файл index.html в наш репозиторий



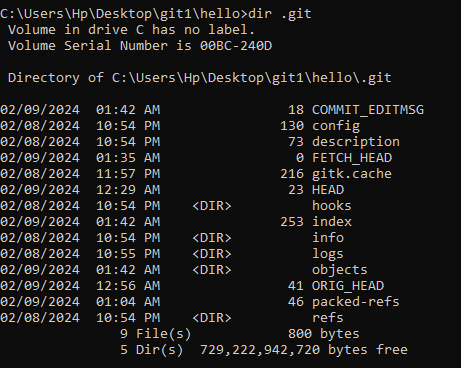
*1.13.1.1.скрипт результат*



*1.13.1.2. Добавьте файл(git add index.html) и сделайте коммит(git commit -m "Added index.html.").*

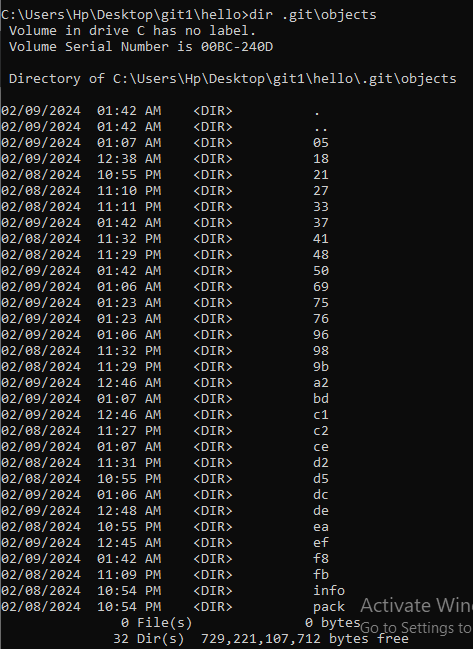
### 1.14 Git внутри: Каталог .git

#### 1.14.1 Каталог .git



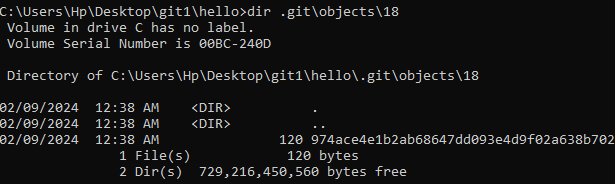
*1.14.1.1.Это каталог, в котором хранится вся информация git*

#### 1.14.2 База данных объектов



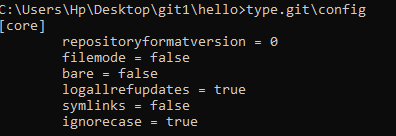
*1.14.2.1.увидеть набор каталогов, имена которых состоят из 2 символов*

### 1.14.3 Углубляемся в базу данных объектов



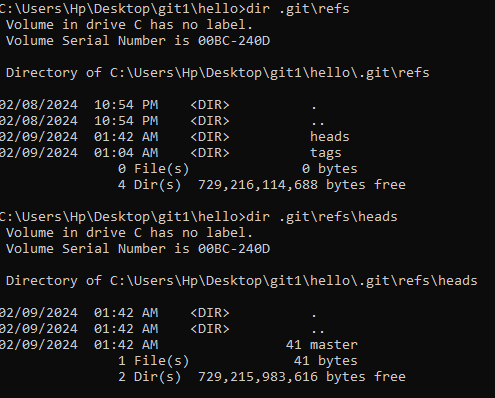
*1.14.3.1.dir .git\objects\18 Смотрим в один из каталогов с именем из 2 букв. Вы увидите файлы с именами из 38 символов*

#### 1.14.4 Config File

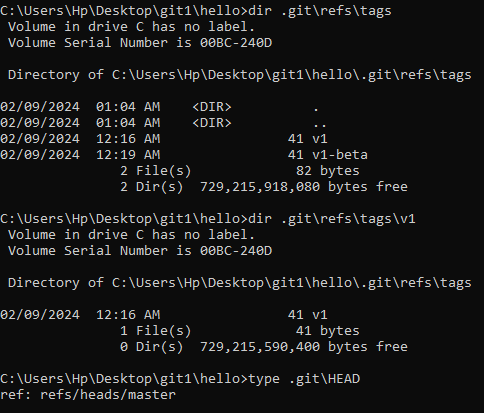


*1.14.4.1.(type .git\config )Это файл конфигурации, создающийся для каждого конкретного проекта*

#### 1.14.5 Ветки и теги



*1.14.5.1.( dir .git\refs, dir .git\refs\heads, dir .git\refs\tags, type .git\refs\tags\v1)*



*1.14.5.2.( dir .git\refs\tags, type .git\refs\tags\v1)*

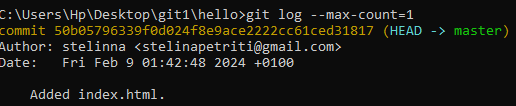
#### 1.14.6 Файл HEAD

fig:

*1.14.6.1.(type .git\HEAD )Файл HEAD содержит ссылку на текущую ветку.*

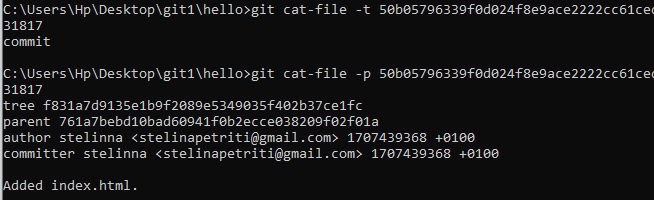
### 1.15 Работа непосредственно с объектами git

#### 1.15.1 Поиск последнего коммита



*1.15.1.1.(git log --max-count=1)Эта команда показать последний коммит в репозиторий*

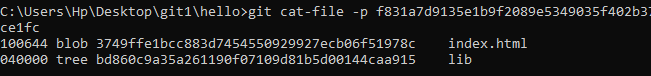
#### 1.15.2 Вывод последнего коммита с помощью SHA1 хэша



*1.15.2.1.(git cat-file -t , git cat-file -p*  *)*

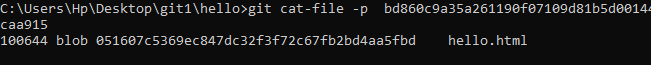
#### 1.15.3 Поиск дерева

Используйте SHA1 хэш из строки «дерева», из списка выше.



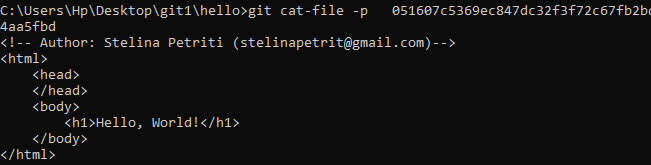
*1.15.3.1.(git cat-file -p )*

#### 1.15.4 Вывод каталога lib



*1.15.4.1.(git cat-file -p )*

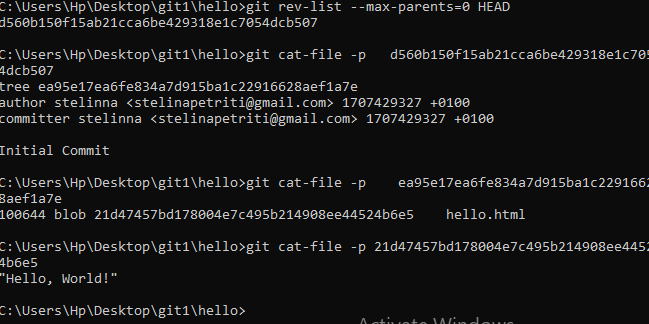
#### 1.15.5 Вывод файла hello.html



*1.15.5.1.(git cat-file -p)*

#### 1.15.6 Исследуйте самостоятельно

Исследуйте git репозиторий вручную самостоятельно. Смотрите, удастся ли вам  
найти оригинальный файл hello.html с самого первого коммита вручную по ссылкам SHA1 хэша в последнем коммите.

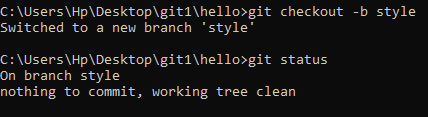


### 1.16 Создание ветки

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.

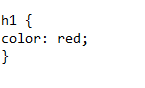
#### 1.16.1 Создайте ветку

Давайте назовем нашу новую ветку «style».

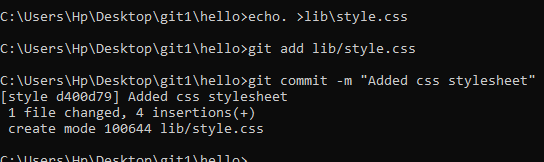


*1.16.1.1.нашу новую ветку «style»(git checkout -b style, git status )*

#### 1.16.2 Добавьте файл стилей style.css



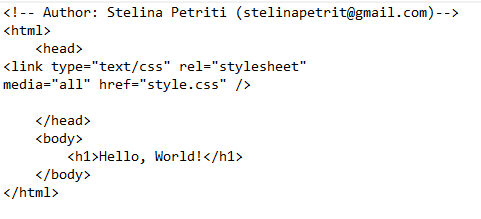
*1.16.2.1.скрипт*



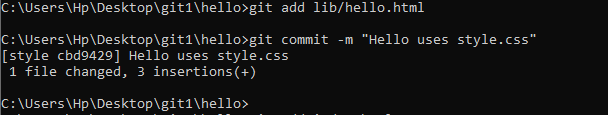
*1.16.2.2.(echo .lib\style.css ,git add lib/style.css, git commit -m "Added css stylesheet" )*

#### 1.16.3 Измените основную страницу

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



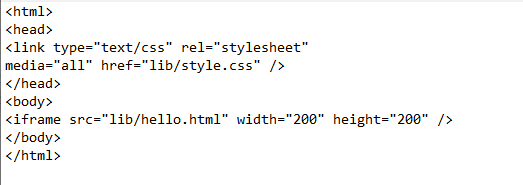
*1.16.3.1.скрипт*



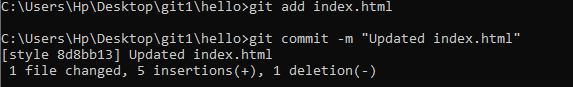
*1.16.3.2.(git add lib/hello.html, git commit -m "Hello uses style.css" )*

#### 1.16.4 Измените index.html

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



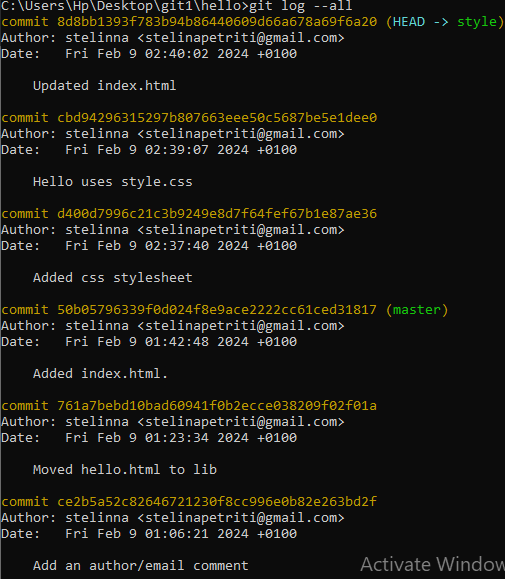
*1.16.4.1.скрипт*



*1.16.4.2.(git add index.html, git commit -m "Updated index.html" )*

### 1.17 Навигация по веткам

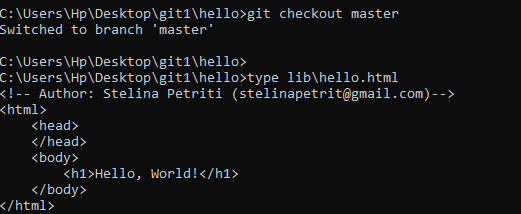
Теперь в вашем проекте есть две ветки:



*1.17.1.(git log --all )*

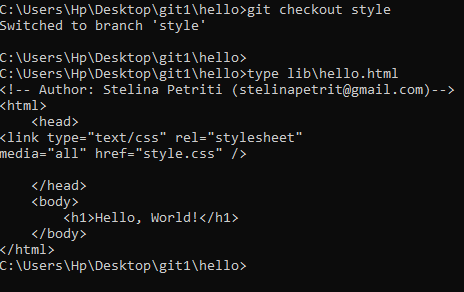
#### 1.17.1 Переключение на ветку master

Используйте команду git checkout для переключения между ветками:



*1.17.1.1.(git checkout master, type lib\hello.html)*

#### 1.17.2 Вернемся к ветке style

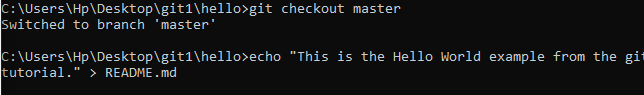


*1.17.2.1.(git checkout style, type lib\hello.html)*

### 1.18 Изменения в ветке master

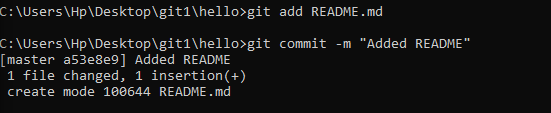
#### 1.18.1 Создайте файл README в ветке master

Создайте файл README.md



*1.18.1.1.(git checkout master,echo "This is the Hello World example from the git tutorial." > README.md )*

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



*1.19.1.(git add README.md, git commit -m "Added README" )*

#### 1.19.1 Просмотр отличающихся веток

#### 1.19.2 Просмотрите текущие ветки

Теперь у нас в репозитории есть две отличающиеся ветки. Используйте следующую  
лог-команду для просмотра веток и их отличий.

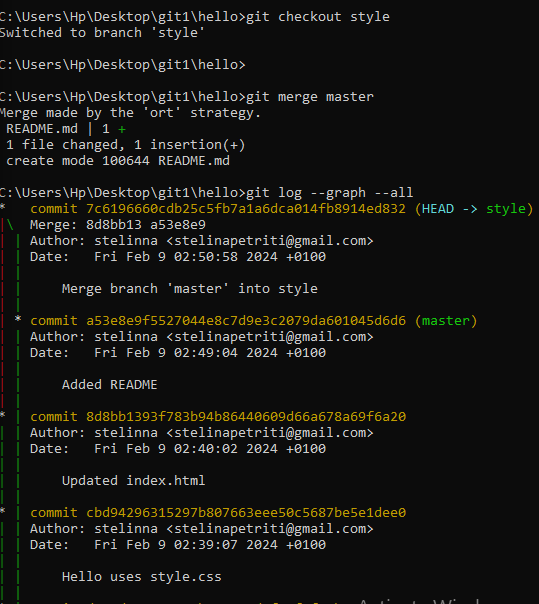


*1.19.2.1.просмотра веток и их отличий(git log --graph --all)*

### 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.

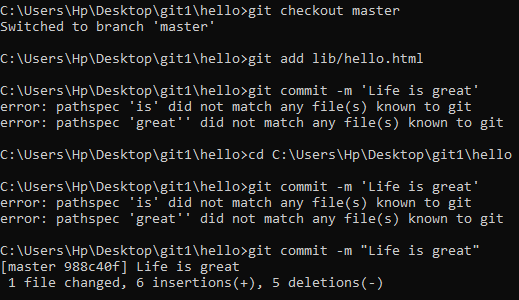


*1.20.1.1.вернемся к ветке style и сольем master с style*

### 1.21 Создание конфликта

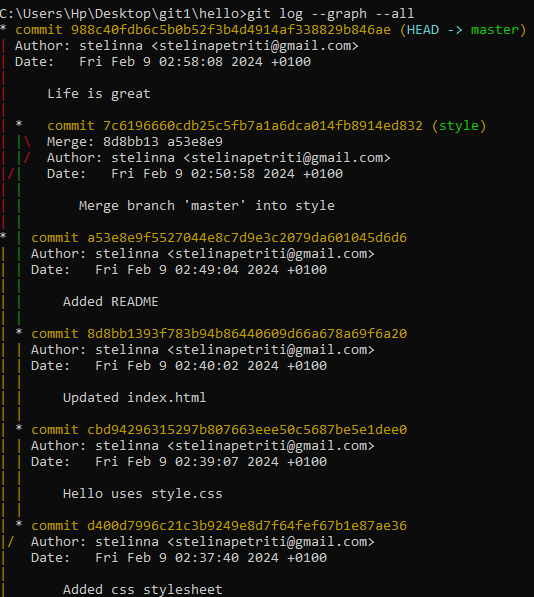
#### 1.21.1 Вернитесь в master и создайте конфликт

Вернитесь в ветку master и внесите следующие изменения:



*1.21.1.1.(git checkout master,git add lib/hello.html, git commit -m 'Life is great' )*

#### 1.21.2 Просмотр веток

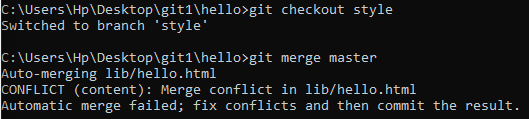


*1.21.2.1.(git log --graph --all)*

### 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.



*1.22.1.1.(git checkout style, git merge master)*

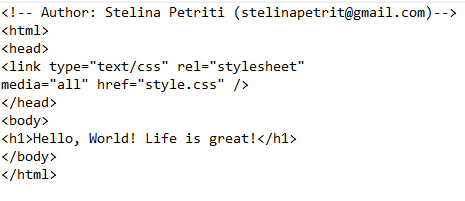
Если вы откроете lib/hello.html, вы увидите:



*1.22.1.2. lib/hello.html, Первый раздел — версия текущей ветки (style). Второй раздел — версия ветки master.*

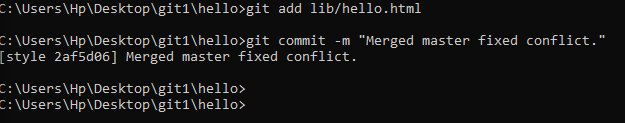
#### 1.22.2 Решение конфликта

Внесите изменения в lib/hello.html для достижения следующего результата.



*1.22.2.1.измените скрипт*

#### 1.22.3 Сделайте коммит решения конфликта

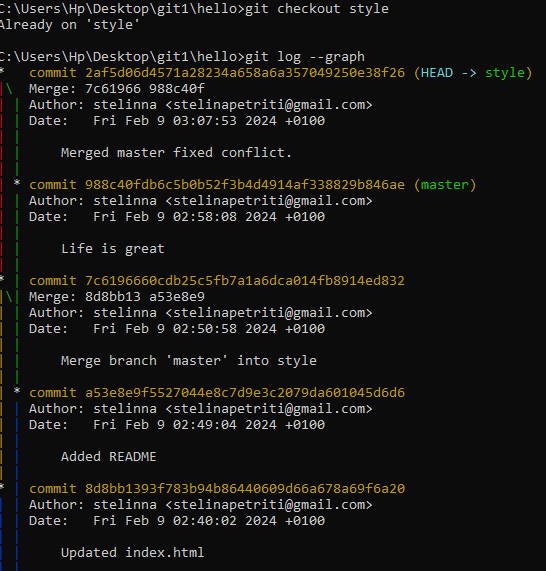


*1.22.3.1.(git add lib/hello.html. git commit -m "Merged master fixed conflict." )*

### 1.23 Сброс ветки style

#### 1.23.1 Сброс ветки style

В этом случае мы хотим вернуться в ветке style в точку перед слиянием с master.  
Нам необходимо найти последний коммит перед слиянием



*1.23.1.1.(git checkout style, git log --graph)*

Давайте сбросим ветку style к этому коммиту.

fig:

*1.23.1.2.(git reset --hard )*

#### 1.23.2 Проверьте ветку.

Поищите лог ветки style. У нас в истории больше нет коммитов слияний.

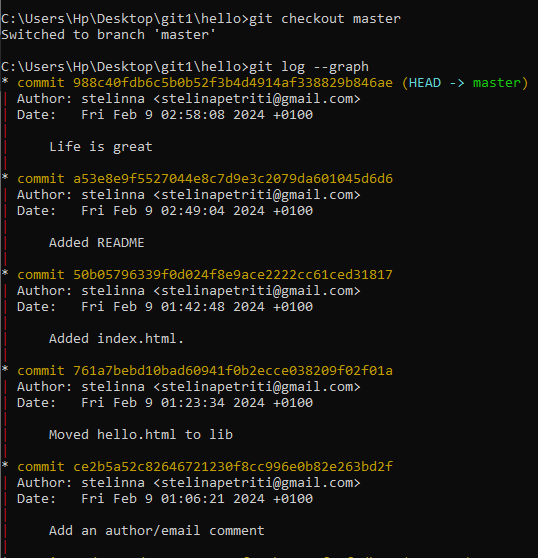


*1.23.2.1.(git log --graph --all)*

### 1.24 Сброс ветки master

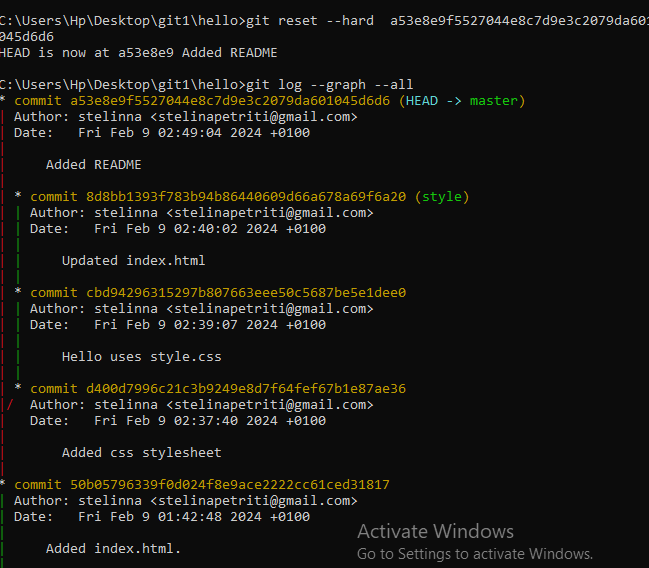
#### 1.24.1 Сброс ветки master

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. Это позволяет нам продемонстрировать работу команды git rebase, не беспокоясь о конфликтах.



*1.24.1.1.вернемся в ветке master в точку перед внесением конфликтующих изменений(git checkout master )*

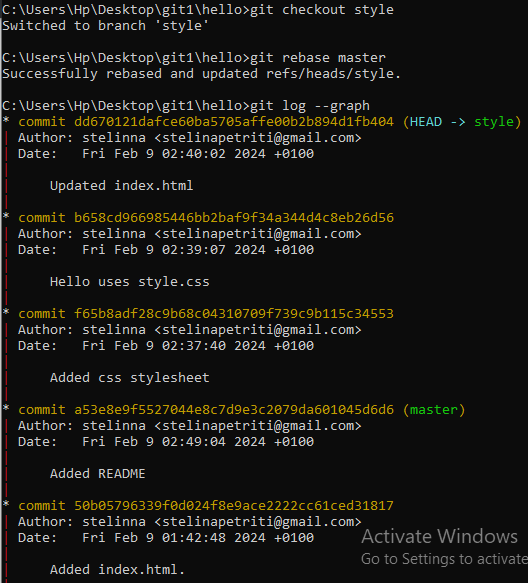
Коммит «Added README» идет непосредственно перед коммитом конфликтующего интерактивного режима. Мы сбросим ветку master к коммиту «Added README».



*1.24.1.2.(git reset --hard , git log --graph --all)*

### 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.

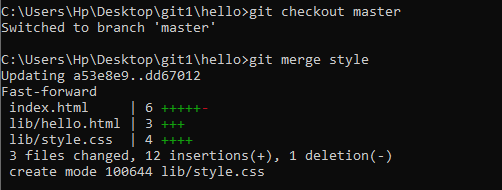


*1.25.1.(git checkout style, git rebase master, git log --graph)*

### 1.26 Слияние в ветку master

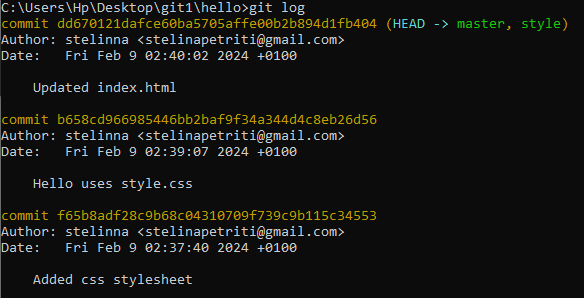
Мы поддерживали соответствие ветки style с веткой master (с помощью rebase),  
теперь давайте сольем изменения style в ветку master.

#### 1.26.1 Слияние style в master



*1.26.1.1.(git checkout master, git merge style)*

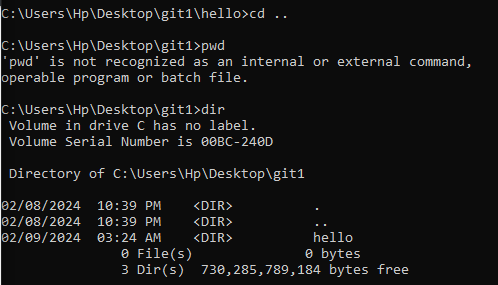
#### 1.26.2 Просмотрите логи



*1.26.2.1.ветки style и master идентичны*

### 1.27 Клонирование репозиториев

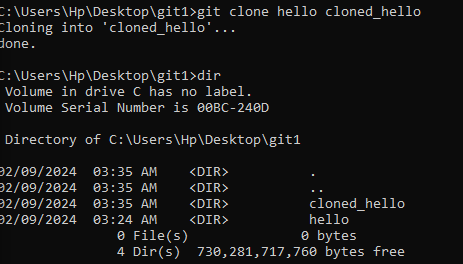
#### 1.27.1 Перейдите в рабочий каталог

Перейдите в рабочий каталог и сделайте клон вашего репозитория hello.  


*1.27.1.(cd .., pwd,dir)\*\*Сейчас мы находимся в рабочем каталоге*

#### 1.27.2 Создайте клон репозитория hello

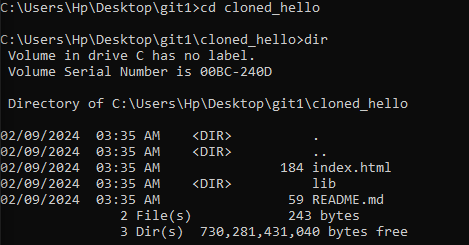
Создадим клон репозитория



*1.27.2.1.(git clone hello cloned\_hello, dir)*

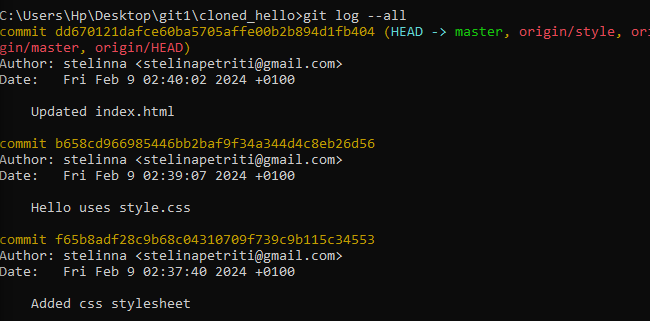
### 1.28 Просмотр клонированного репозитория

#### 1.28.1 Давайте взглянем на клонированный репозиторий.



*1.28.1.1.(cd cloned\_hello, dir)*

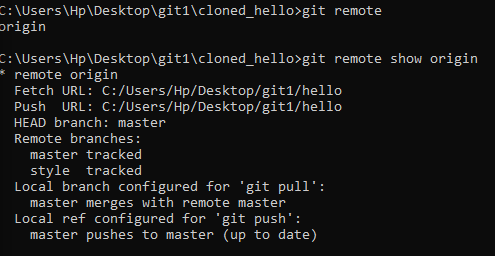
#### 1.28.2 Просмотрите историю репозитория



*1.28.2.1.(git log --all)*

### 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:



*1.29.1.(git remote,git remote show origin)*

### 1.30 Удаленные ветки

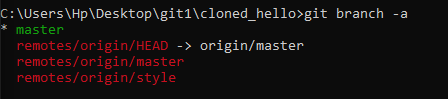
Давайте посмотрим на ветки, доступные в нашем клонированном репозитории.

fig:

1.30.1.(git branch)

#### 1.30.1 Список удаленных веток

Для того, чтобы увидеть все ветки:



*1.30.1.1.(git branch -a)*

### 1.31 Изменение оригинального репозитория

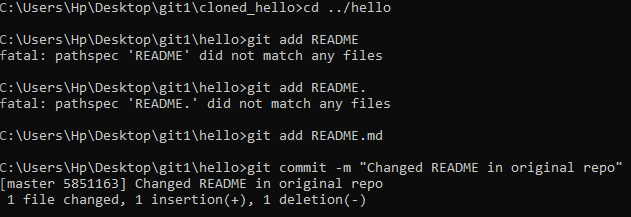
Внесите некоторые изменения в оригинальный репозиторий, чтобы затем попытаться извлечь и слить изменения из удаленной ветки в текущую

#### 1.31.1 Внесите изменения в оригинальный репозиторий hello

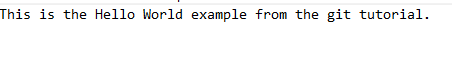
Выполните:  
cd ../hello

Примечание: Сейчас мы находимся в репозитории hello  
Внесите следующие изменения в файл README.md:  
Файл README.md

This is the Hello World example from the git tutorial.  
Теперь добавьте это изменение и сделайте коммит  
Выполните:  
git add README  
git commit -m "Changed README in original repo"



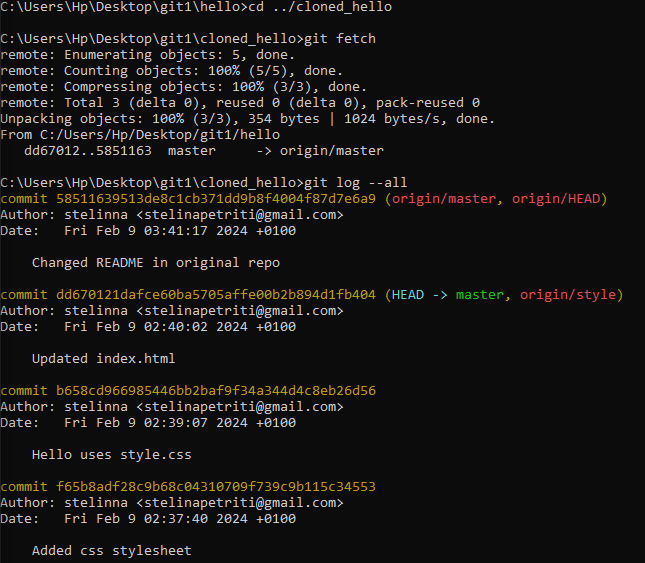
*1.31.1.1.(cd ../hello,git add README, git commit -m "Changed README in original repo")*



*1.31.1.2.( изменения )*

#### 1.31.2 Извлечение изменений

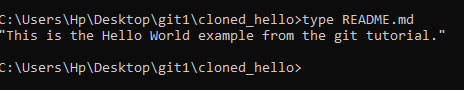
Научиться извлекать изменения из удаленного репозитория.



*1.31.2.1.(cd ../cloned\_hello, git fetch, git log --all )*

#### 1.31.3 Проверьте README.md

Мы можем продемонстрировать, что клонированный файл README.md не изменился.



*1.31.3.1.(type README)*

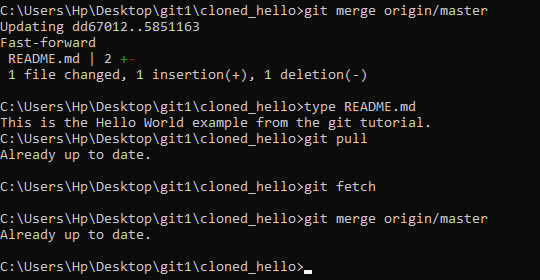
### 1.32 Слияние извлеченных изменений

#### 1.32.1 Слейте извлеченные изменения в локальную ветку master

Выполните:  
git merge origin/master

#### 1.32.2 Еще раз проверьте файл README.md

Сейчас мы должны увидеть изменения.  
Выполните:  
cat README.md  
Хотя команда git fetch не сливает изменения, мы можем вручную слить изменения из удаленного репозитория.  
Теперь давайте рассмотрим объединение fetch и merge в одну команду. Выполнение:  
git pull  
эквивалентно двум следующим шагам:  
git fetch  
git merge origin/master



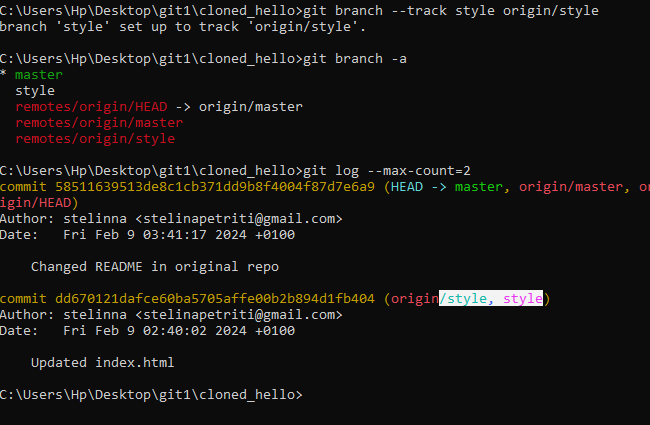
*1.32.1.2.(Слейте извлеченные изменения в локальную ветку master, проверьте файл README.md)*

### 1.33 Добавление ветки наблюдения

Ветки, которые начинаются с remotes/origin являются ветками оригинального  
репозитория.Обратите внимание,что у вас больше нет ветки под названием style,  
но система контроля версий знает, что в оригинальном репозитории ветка style  
была.

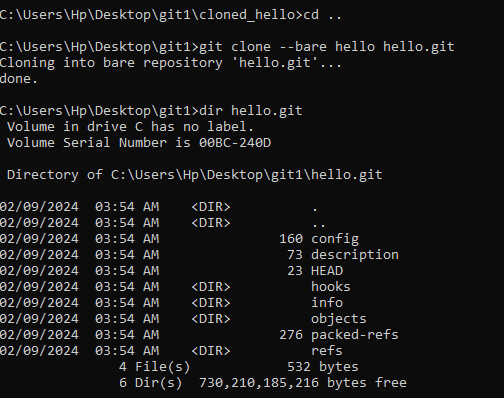
#### 1.33.1 Добавьте локальную ветку, которая отслеживает удаленную ветку

Выполните:  
git branch --track style origin/style  
git branch -a  
git log --max-count=2  
Теперь мы можем видеть ветку style в списке веток и логе.



*1.33.1.1.Добавьте локальную ветку*

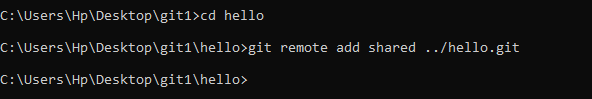
### 1.35 Создайте чистый репозиторий



*1.35.1.(cd .., git clone --bare hello hello.git, dir hello.git )*

### 1.36 Добавление удаленного репозитория

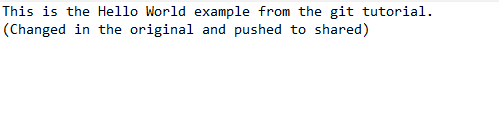
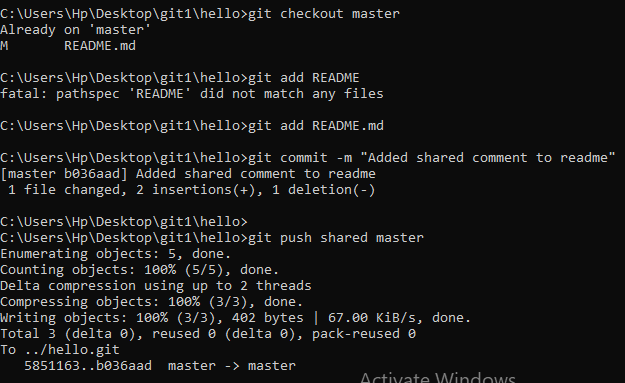
Давайте добавим репозиторий hello.git к нашему оригинальному репозиторию.



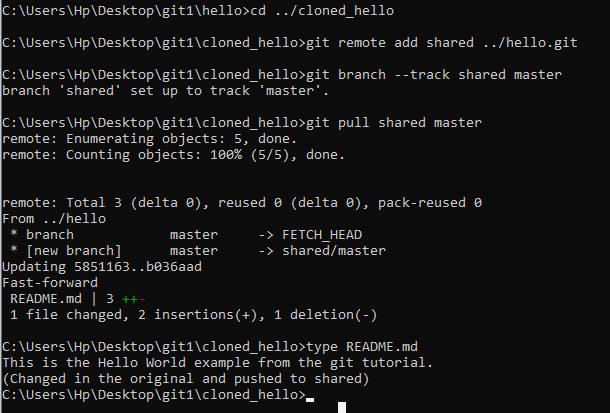
*1.36.1.(cd hello, git remote add shared ../hello.git )*

### 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:  
*1.37.1.This is the Hello World example from the git tutorial.*  
(Changed in the original and pushed to shared)  
\*\*  
1.37.2.(git checkout master, git add README, git commit -m "Added shared comment to readme",git push shared master)  
Теперь отправьте изменения в общий репозиторий.  
Общим называется репозиторий, получающий отправленные нами изменения.

### 1.38 Извлечение общих изменений

Быстро переключитесь в клонированный репозиторий и извлеките изменения, только что отправленные в общий репозиторий.  
  
*\*1.38.1.cd ../cloned\_hello*  
*Сейчас мы находимся в репозитории cloned\_hello.*  
*git remote add shared ../hello.git*  
*git branch --track shared master*  
*git pull shared master*  
cat README.md\*

# Вывод

B этой лабораторной работе мы осваиваем навыки работы с git. Решение проблем, создание, добавление, удаление, изменение и другие важные команды, которые необходимы для овладения навыками работы с git.