# \_УП\_1

# Работа с Git

## Подготовка

Если git никогда ранее не использовался пользователем, то необходимо установить имя и электронную почту. Для этого требуется выполнить команды, показанные на рисунке 1.

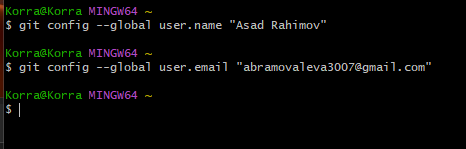


Рисунок 1 - Установка имени и электронной почты

Затем нужно указать параметры установки окончаний строк (рисунок 2).

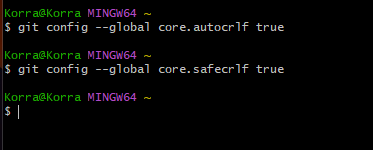


Рисунок 2 - Параметры окончаний строк

И последним пунктом идет установка отображения Unicode, показанная на рисунке 3.

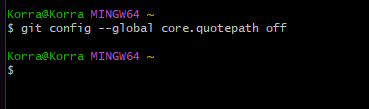


Рисунок 3 - Установка отображения unicode

## Создание проекта

Сначала нужно создать пустой каталог и внутри него файл hello.html (рисунок 4).

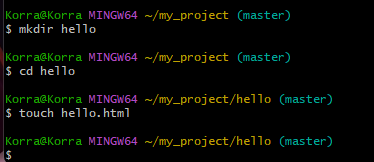


Рисунок 4 - Создание каталога и файла

После этого в файл необходимо ввести данные, например, «Hello, world», как показано на рисунке 5.

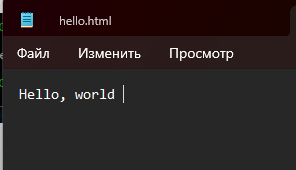
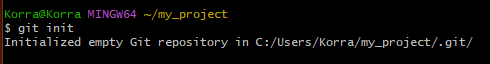


Рисунок 5 - Содержание файла

Для создания репозитория

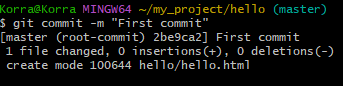


используется команда git init (рисунок 6).

Рисунок 6 - Создание репозитория

Для добавления страницы в репозиторий необходима команда git add (рисунок 7).

Рисунок 7 - Добавление в репозиторий



## Проверка состояния

Проверка состояния репозитория осуществляется с помощью команды git status. Если в репозитории хранится текущее состояние рабочего каталога и нет изменений, ожидающих записи, будет показано сообщение, как на рисунке 8.

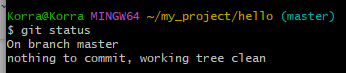


Рисунок 8 - Проверка состояния репозитория

## Внесение изменений

Сначала необходимо внести изменения в файл (рисунок 9).

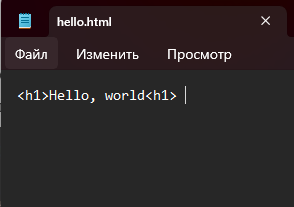
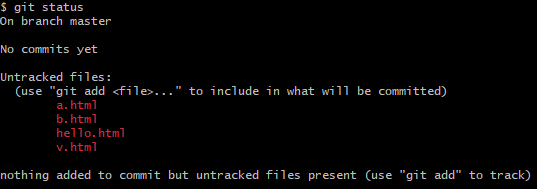


Рисунок 9 - Внесение изменений в файл

Если после предыдущего пункта осуществить проверку состояния репозитория, то будет показано данное сообщение (рисунок 10).

Рисунок 10 - Сообщение о незафиксированных изменениях



## Индексация изменений

Для того, чтобы проиндексировать изменения, нужно осуществить действия, показанные на рисунке 11.

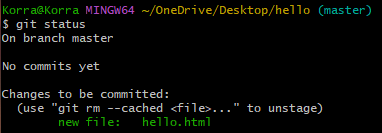


Рисунок 11 - Команды для индексации изменений

После этого изменения файлы были проиндексированы. Это значит, что пока изменения не записаны в репозиторий. Если изменения позже не нужно будет фиксировать, то индексацию можно снять командой git reset.

## Индексация и коммит

Можно зафиксировать изменения отдельными коммитами. Как это сделать, показано на рисунках 12-14.

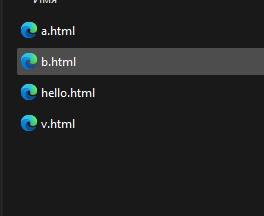


Рисунок 12 - Создано 3 файла

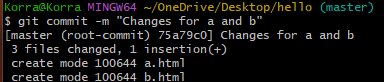


Рисунок 13 - Индексация и коммит для 2 файлов

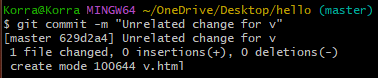


Рисунок 14 - Индексация и коммит для третьего файла

## Коммит изменений

Для того, чтобы редактировать комментарий коммита, нужно использовать команду git commit без метки -m.

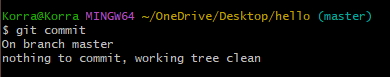


Рисунок 15 - Коммит изменений

После этого будет открыт редактор, в котором необходимо на 1 строке записать комментарий коммита (рисунок 16).

Рисунок 16 - Ввод комментария

После выхода из текстового редактора будет указано следующее сообщение (рисунок 17).

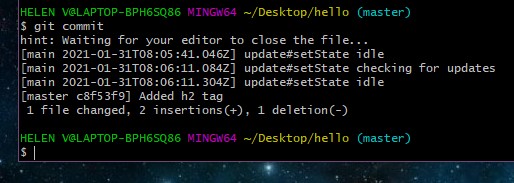


Рисунок 17 - Коммит-сообщения

После этого еще раз нужно проверить состояние репозитория (рисунок 18).

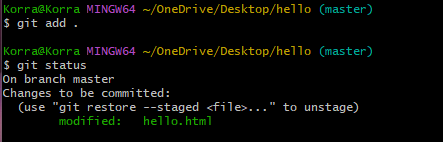


Рисунок 18 - Проверка состояния

## Изменения, а не файлы

Для того, чтобы понять, что git фокусируется на изменениях в файле, а не на самом файле, можно проделать следующие действия.

Сначала нужно в файл hello.html добавить теги html и body (рисунок 19), а затем проиндексировать изменения.

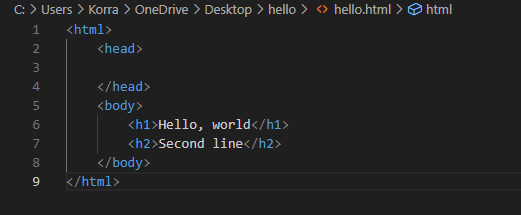


Рисунок 19 - Добавление тегов html и body

Затем еще раз нужно добавить изменения в файл (добавить тег head), но изменения не индексировать (рисунок 20).

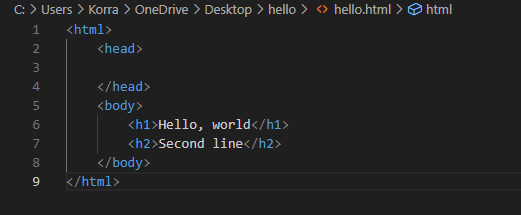


Рисунок 20 - Добавление тега head

Далее нужно проверить статус. На рисунке 21 видно, что файл hello.html указан дважды: первое изменение проиндексировано и готово к коммиту, а второе – нет.

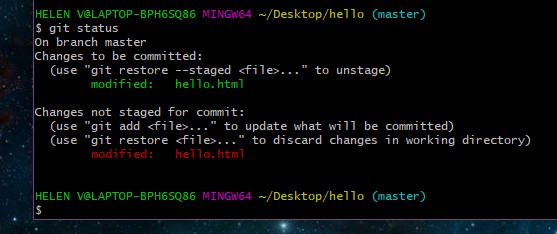


Рисунок 21 - Проверка состояния

Далее надо произвести коммит проиндексированного изменения и затем еще раз проверить состояние (рисунок 22).

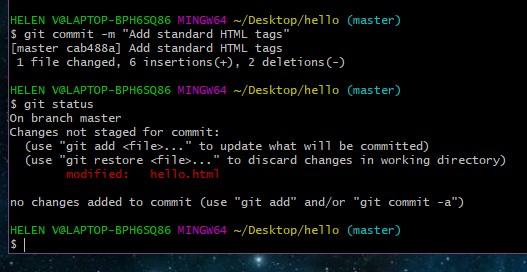


Рисунок 22 - Коммит 1 изменения и проверка состояния

Нужно добавить второе изменение в индекс и затем проверить состояние (рисунок 23).

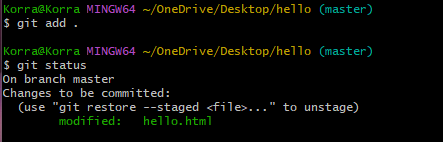


Рисунок 23 - Индексация 2 изменения и проверка состояния

После этого нужно сделать коммит второго изменения (рисунок 24).

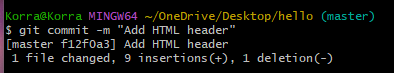


Рисунок 24 - Коммит 2 изменения

# \_УП\_2

# Работа с Git

## История

Для того, чтобы просмотреть список произведенных изменений в проекте, используется команда git log (рисунок 1).

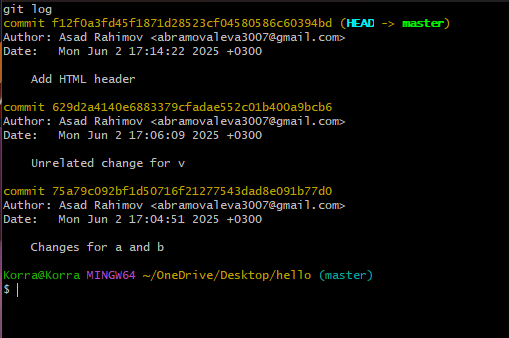


Рисунок 25 - Просмотр истории изменений

На рисунке 1 была выведена полная история. Для того, чтобы увидеть однострочный формат используется команда git log –pretty=oneline (рисунок 2).

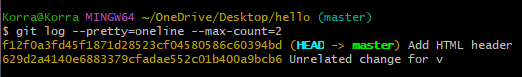


Рисунок 26 - Однострочный формат вывода

Далее на рисунках 3-8 показано несколько вариантов вывода истории изменений.

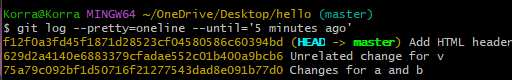


Рисунок 27 - Вывод последних 2 изменений

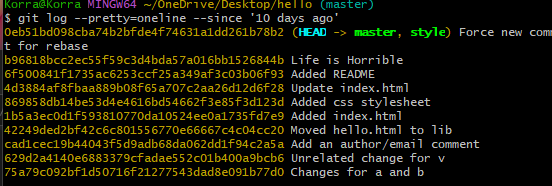


Рисунок 28 - Вывод изменений начиная с определенного времени

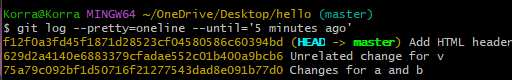


Рисунок 29 - Вывод изменений до определенного времени

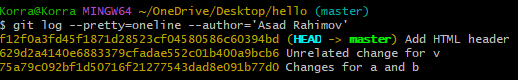


Рисунок 30 - Вывод изменений, внесенных определенным автором

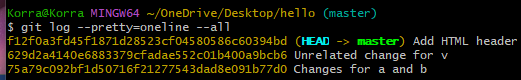


Рисунок 31 - Вывод всех изменений

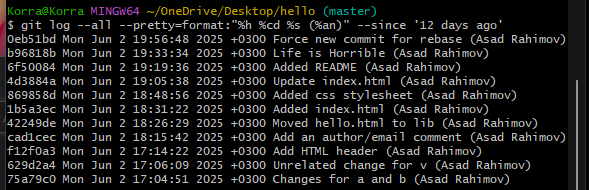


Рисунок 32 - Использование нескольких параметров

## Алиасы

Для настройки алиасов используется команда, показанная на рисунке 9.



Рисунок 33 - Настройка алиасов для некоторых команд

При выполнении алиаса будет выполнена определенная команда и выведены нужные данные (рисунок 10).

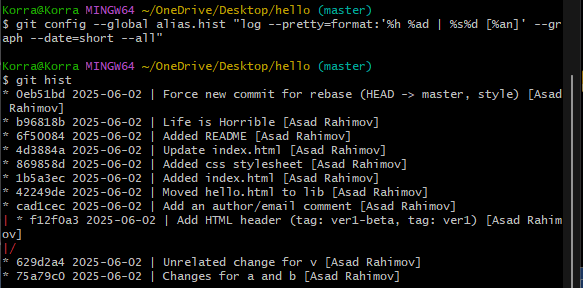


Рисунок 34 - Выполнение алиаса hist

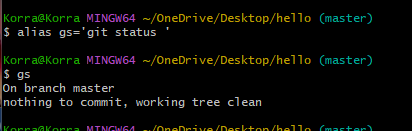


Рисунок 35 - Установка и выполнение алиаса gs

## Получение старых версий

Для того, чтобы вернуть рабочий каталог к предыдущему состоянию, можно использовать следующий способ: для начала нужно узнать хэши предыдущих версий, что можно сделать с помощью ранее заданного алиаса hist (рисунок 12).

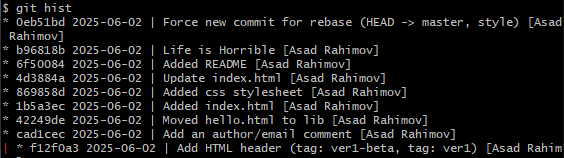


Рисунок 36 - Просмотр хэшей предыдущих версий

Далее нужно выполнить команду git checkout с номером нужного хэша (достаточно первых 7 знаков). После этого можно просмотреть содержимое файла с помощью команды cat (рисунок 13).



Рисунок 37 - Возвращение к нужной версии и просмотр содержимого файла

## Возвращение к последней версии в ветке master

Для возвращения к последней версии в ветке master (имя ветки по умолчанию) надо ввести команду git checkout master, что показано на рисунке 14.



Рисунок 38 - Возвращение к последней версии в ветке master

## Создание тегов версий

Для создания тега используется команда git tag. На рисунке 15 показано, тегом ver1 была названа текущая версия страницы.



Рисунок 39 - Задание тега

Чтобы перейти к предыдущей версии, можно использовать символ «^», который означает «родитель».

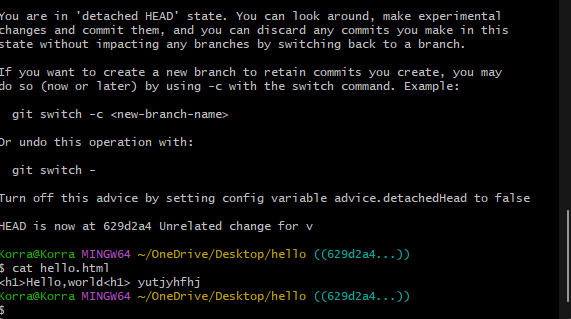


Рисунок 40 – Переход к предыдущей версии с помощью тега



Рисунок 41 - Задание тега предыдущей версии

Теперь с помощью тегов можно переключаться между версиями (рисунок 18).

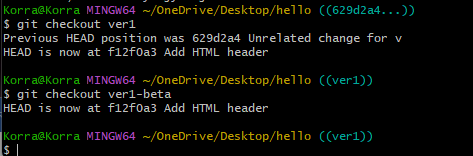


Рисунок 42 - Переключение между версиями с помощью тегов

Для просмотра всех тегов используется команда git tag (рисунок 19).



Рисунок 43 - Просмотр тегов

Также можно просмотреть теги в логе, как показано на рисунке 20.

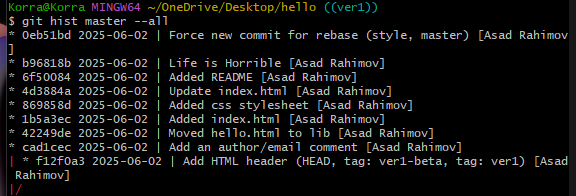


Рисунок 44 - Просмотр тегов в логе

## Отмена локальных изменений (до индексации)

Сначала нужно переключиться на последний коммит master (рисунок 21).

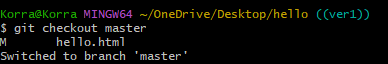


Рисунок 45 - Переключение на последний коммит

Далее для работы нужно внести изменение в файл (рисунок 22).

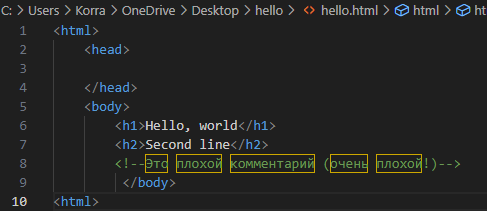


Рисунок 46 - Внесение изменения в файл

После выполнения команды git status будет показано, что есть не проиндексированное изменение (рисунок 23).

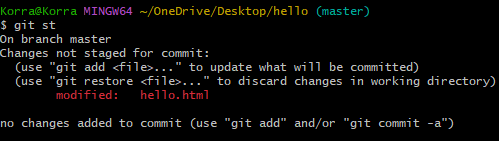


Рисунок 47 - Не проиндексированное изменение

Для переключения в версию файла без изменений используется команда git checkout hello.html (рисунок 24). Команда git status покажет, что не было произведено изменений, не зафиксированных в каталоге.

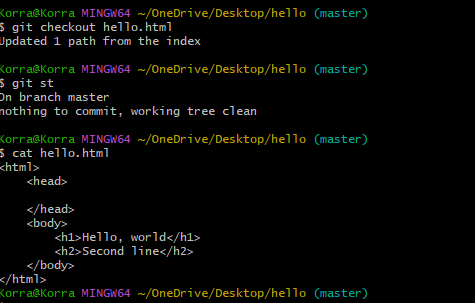
****

Рисунок 48 - Возвращение к версии

## Отмена проиндексированных изменений (перед коммитом)

Для того, чтобы научиться отменять проиндексированные изменения, сначала нужно внести ненужное изменение в файл (рисунок 25). После этого производится индексация (рисунок 26).

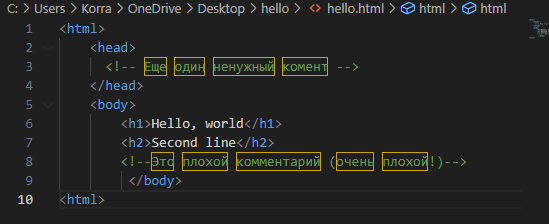
****

Рисунок 49 - Внесение ненужного изменения

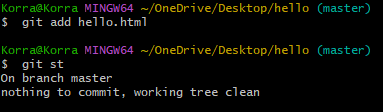
****

Рисунок 50 - Индексация изменения

Для отмены индексация изменения используется команда git reset HEAD hello.html (рисунок 27). Команда reset сбрасывает буферную зону к HEAD и очищает ее от проиндексированных изменений. Но для удаления ненужного по-прежнему используется команда git checkout (рисунок 28).

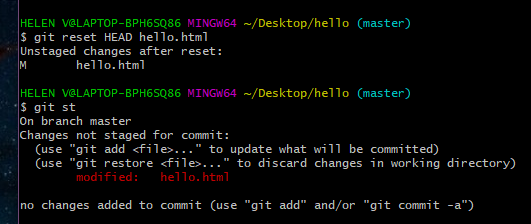


Рисунок 51 - Очистка буферной зоны

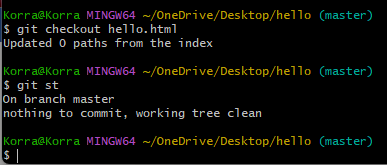
****

Рисунок 52 - Удаление ненужных изменений

## Отмена коммитов

Для отмены коммита можно использовать способ создания нового коммита, отменяющего изменения.

Для начала надо внести изменение, проиндексировать его и записать коммит (рисунки 29-30).

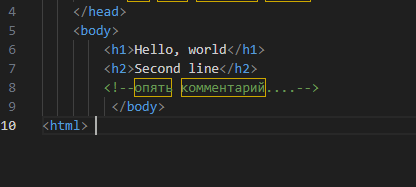
****

Рисунок 53 - Внесение изменения в файл

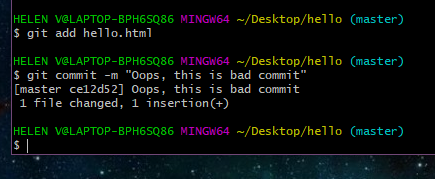


Рисунок 54 - Индексация и коммит

Для создания коммита, который удалит ненужные изменения, используется команда git revert HEAD (рисунок 31). После этого будет открыт редактор, в котором можно отредактировать коммит сообщение (рисунок 32), затем надо сохранить файл и закрыть редактор (рисунок 33).

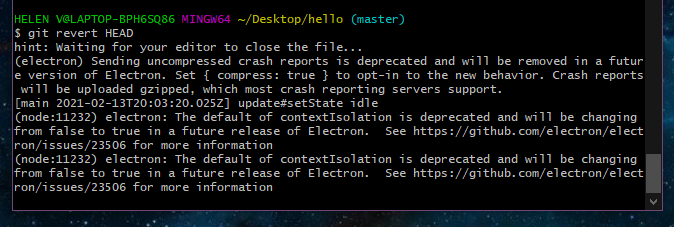


Рисунок 55 - Выполнение команды git revert

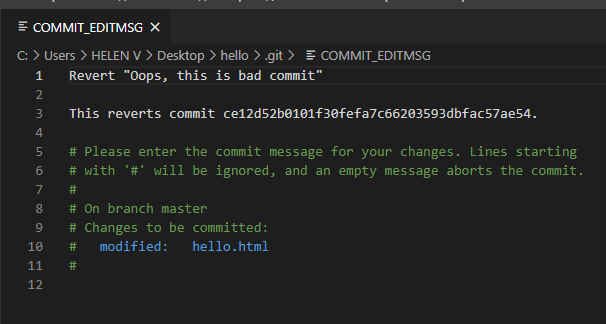


Рисунок 56 - Коммит сообщение в редакторе

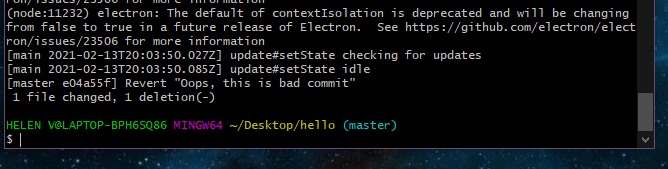


Рисунок 57 - Редактор закрыт

При проверке лога будут показаны все коммиты, в том числе и отмененные (рисунок 34).

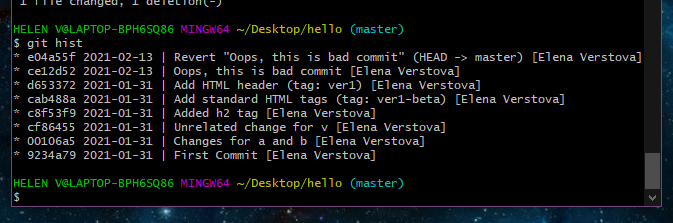


Рисунок 58 - Все коммиты при просмотре лога

Перед удалением коммита последний из них нужно отметить тегом, чтобы не потерять его (рисунок 35).

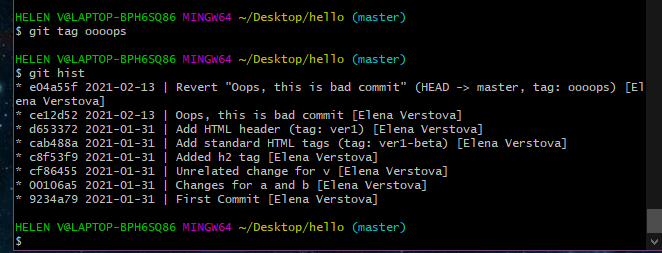


Рисунок 59 - Отметка тегом коммита

Для сброса коммитов используется команда git reset –hard ver1 (рисунок 36). Она сбрасывает ветку до версии с тегом ver1.

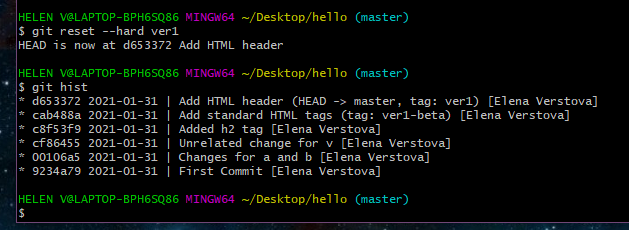


Рисунок 60 - Сброс коммита

Но при просмотре лога с помощью команды git hist –all отмененные коммиты по-прежнему будут показываться, так как они всё еще находятся в репозитории (рисунок 37).

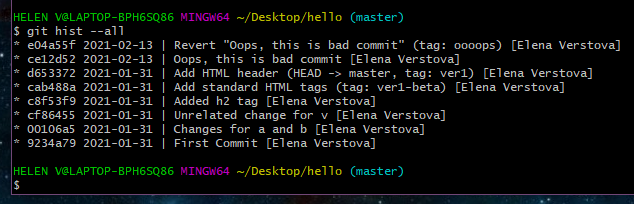


Рисунок 61 - Сброшенные коммиты находятся по-прежнему в репозитории

## Удаление тега

Так как тег «oooops» больше не нужен, его и коммиты, на которые он указывает, можно удалить с помощью команды git tag -d (рисунок 38).

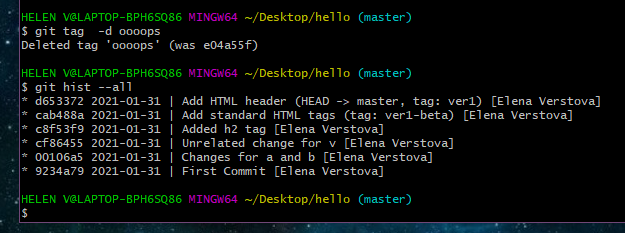


Рисунок 62 - Удаление тега

# \_УП\_3

# Работа с Git

## Внесение изменений в коммиты

Для начала будет создан коммит, в который позже будут внесены изменения. На рисунках 1 и 2 происходит добавление комментария в файл hello.html и его индексация и коммит.

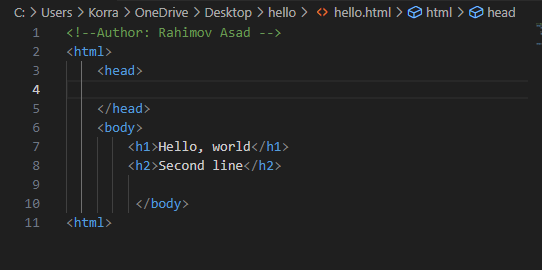
****

Рисунок 63 - Добавление комментария в файл

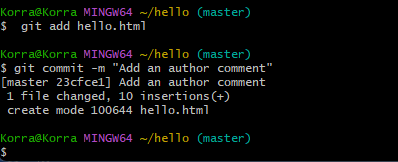


Рисунок 64 - Индексация и коммит

Далее необходимо добавить электронную почту в комментарий (рисунок 3).

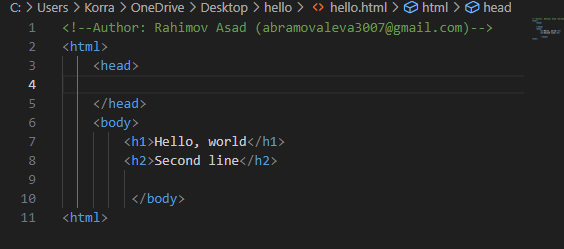
****

Рисунок 65 - Добавление электронной почты

Но для того, чтобы не создавать отдельный коммит ради электронной почты, можно изменить предыдущий так, как показано на рисунке 4.

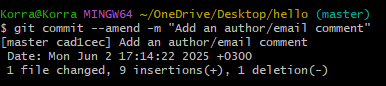
****

Рисунок 66 - Индексация и изменение коммита

При просмотре истории можно будет заметить, что последний коммит был изменен.

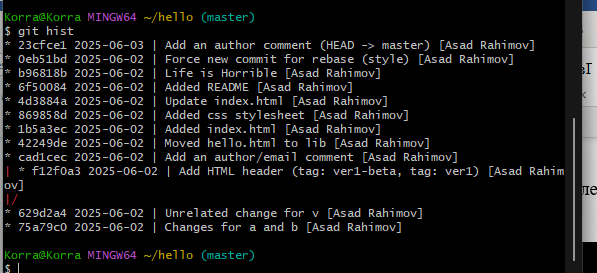


Рисунок 67 - Последний коммит изменен

## Перемещение файлов

Для перемещения файлов в пределах репозитория используются команды, показанные на рисунке 6. После выполнения данных команды git индексирует эти изменения (удаление файла hello.html и создание файла lib/hello.html).

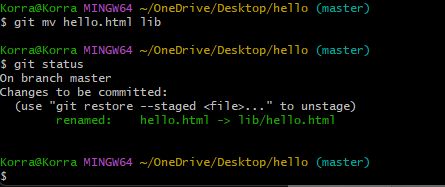
****

Рисунок 68 - Перемещение файла

Далее надо осуществить коммит данного перемещения.

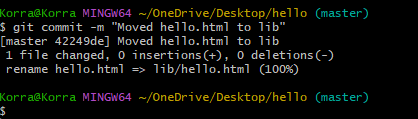
****

Рисунок 69 – Коммит перемещения

## Подробнее о структуре

Необходимо добавить еще один файл в репозиторий. Это будет файл index.html с кодом, показанным на рисунке 8.

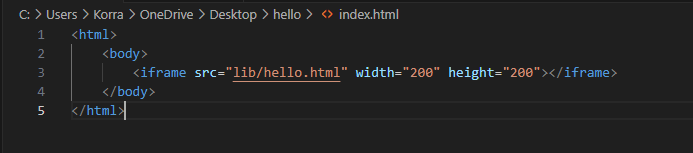
****

Рисунок 70 - Содержимое файла index.html

Далее нужно проиндексировать и закоммитить файл (рисунок 9).

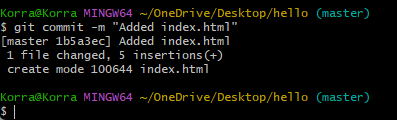
****

Рисунок 71 - Индексация и коммит

При открытии файла index.html будет виден кусок страницы hello.html (рисунок 10).

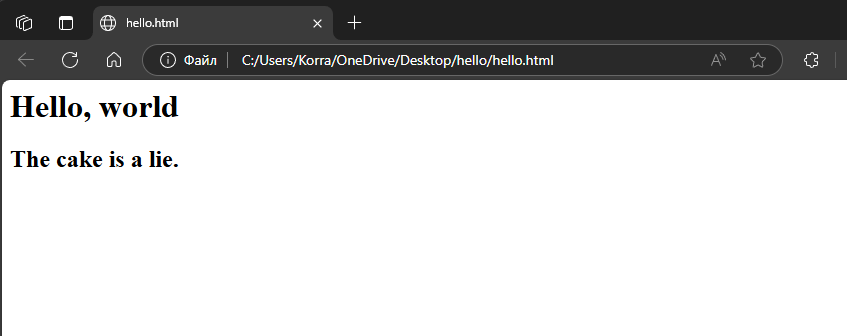
****

Рисунок 72 - Файл index.html, открытый в браузере

## Каталог .git

Чтобы посмотреть структуру каталога .git необходимо выполнить команду, показанную на рисунке 11.

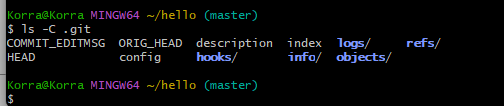


Рисунок 73 - Содержание каталога .git

При аналогичном просмотре каталога objects можно будет увидеть множество каталогов с именами из 2 символов (рисунок 12). Имена каталогов являются первыми двумя буквами хэша.

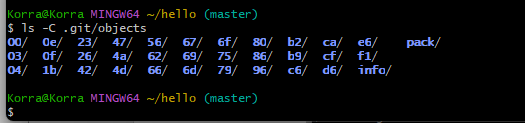


Рисунок 74 - Содержание каталога objects

При просмотре содержимого любого из каталогов будут показаны файлы, названия которых состоят из 38 символов (рисунок 13).

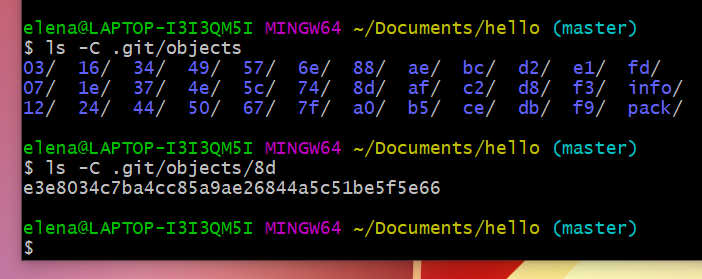


Рисунок 75 - Просмотр каталога 8d

Далее требуется просмотреть файл конфигурации с помощью команды cat (рисунок 14).

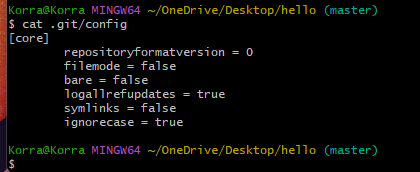
****

Рисунок 76 - Просмотр файла конфигурации

На рисунке 14 показан просмотр файлов в подкаталоге tags и веток в каталоге heads.

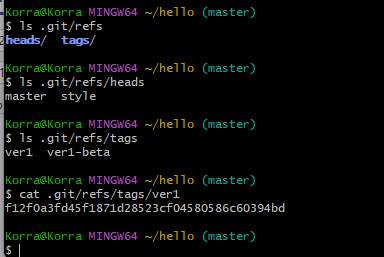


Рисунок 77 - Просмотр файлов и веток

Файл HEAD содержит ссылку на текущую ветку (рисунок 16).

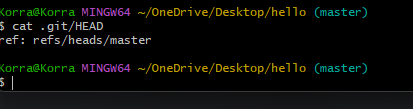
****

Рисунок 78 - Содержимое файла HEAD

## Работа с объектами git

Для начала необходимо просмотреть последний коммит (рисунок 17).

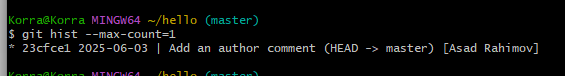


Рисунок 79 - Последний коммит

Далее надо использовать хэш последнего коммита, используя команды cat-file -p и cat-file -t (рисунок18) для просмотра объекта коммита. Также вместо длинных команд можно использовать сокращенные type и dump, если данные команды были заданы как алиасы.



Рисунок 80 - Просмотр объекта коммита

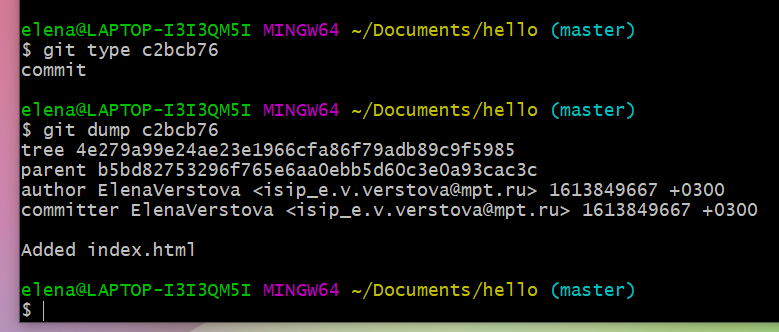


Рисунок 81 - Использование алиасов

Для просмотра дерева каталогов необходимо использовать его хэш (рисунок 20).

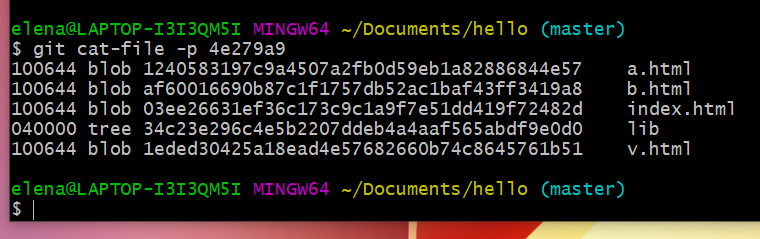


Рисунок 82 - Просмотр дерева каталогов

Затем нужно просмотреть каталог lib (рисунок 21).

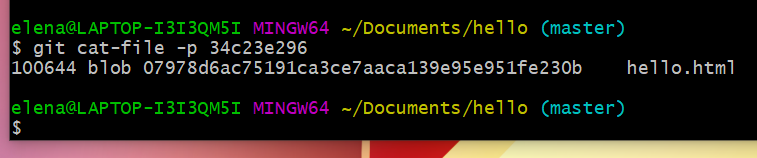


Рисунок 83 - Просмотр каталога lib

И затем требуется вывести содержимое файла hello.html (рисунок 22).

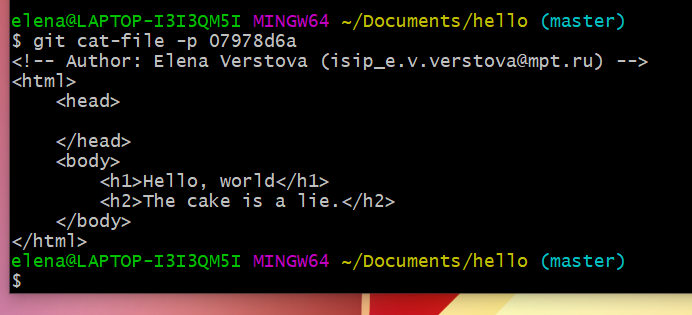


Рисунок 84 - Вывод содержимого файла hello.html

Аналогичным образом можно просмотреть содержимое файла, каким оно было в самом первом коммите, как показано на рисунке 23. Для этого требуется использовать лишь нужные хэши.

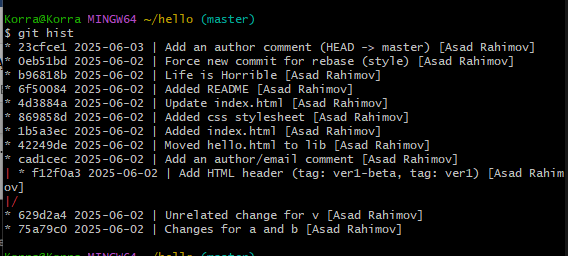


Рисунок 85 - Просмотр содержимого файла при первом коммите

# \_УП\_4

# Работа с Git

## Создание ветки

Для начала необходимо создать ветку style с помощью команды git checkout -b style (рисунок 1).

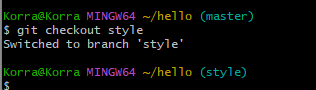


Рисунок 86 - Создание новой ветки style

Затем нужно создать файл стилей (рисунок 2) и внести в него код, показанный на рисунке 3.

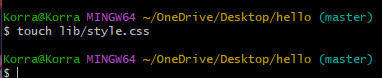
****

Рисунок 87 - Создание файла стилей

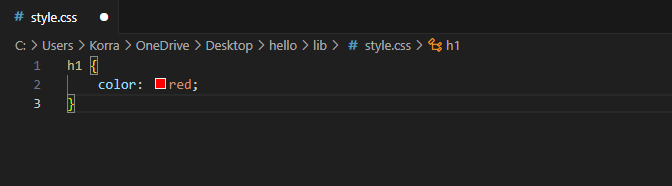
****

Рисунок 88 - Код style.css

После этого надо произвести индексацию и коммит (рисунок 4).

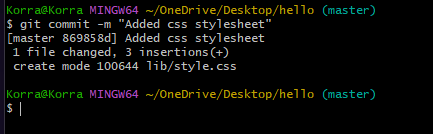


Рисунок 89 - Индексация и коммит нового файла

Далее требуется изменить основную страницу hello.html и закоммитить изменения (рисунки 5-6).

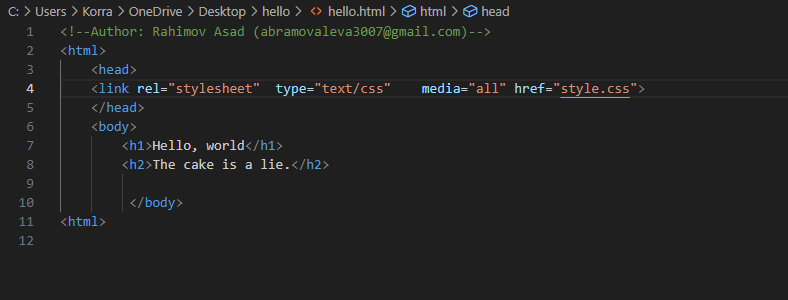


Рисунок 90 - Изменения в файле hello.html

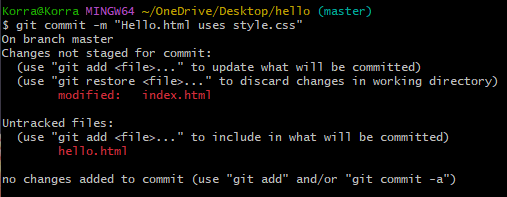


Рисунок 91 - Индексация и коммит

Далее аналогичные действия нужно осуществить с файлом index.html, как это показано на рисунках 7-8.

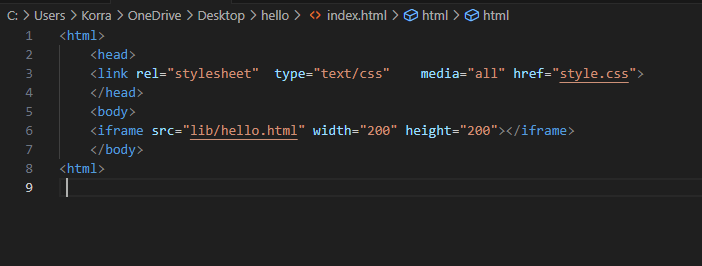


Рисунок 92 - Изменения в файле index.html

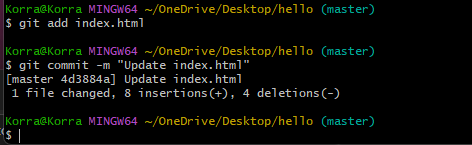


Рисунок 93 - Индексация и коммит

После выполнения предыдущих действий была создана новая ветка style с 3 коммитами.

## Навигация по веткам

При просмотре истории, как на рисунке 9, можно увидеть, что теперь в проекте 2 ветки.

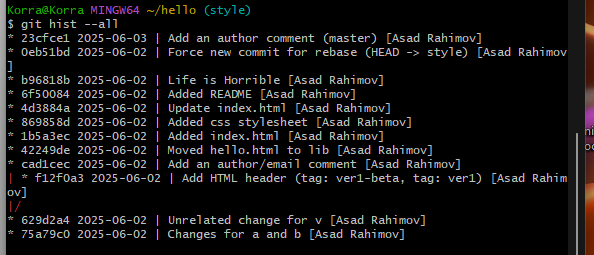


Рисунок 94 - Просмотр истории

Для переключения на ветку master используется команда git checkout master (рисунок 10). После переключения на нужную ветку при выводе файла hello.html можно увидеть, что изменения отсутствуют (по причине того, что они закоммичены в другой ветке).

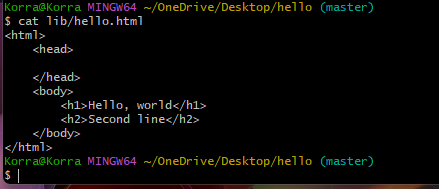


Рисунок 95 - Переключение на ветку master

При переключении на ветку style файл hello.html будет иметь другое содержание (рисунок 11).

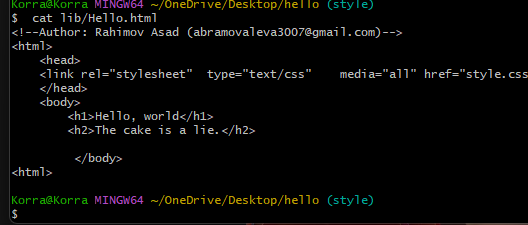


Рисунок 96 - Переключение на ветку style

## Изменения в ветке master

Необходимо переключиться на ветку master (рисунок 12) и добавить файл README (рисунки 13-14).

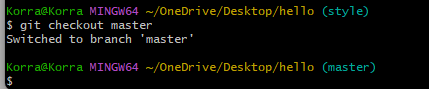


Рисунок 97 - Переключение на ветку master



Рисунок 98 - Содержимое файла README

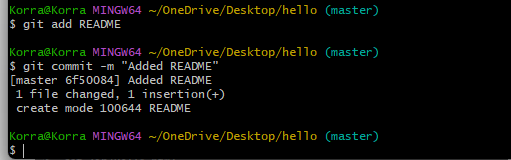


Рисунок 99 - Индексация и коммит

## Просмотр отличающихся веток

На рисунке 15 можно увидеть дерево коммитов.



Рисунок 100 - Дерево коммитов

## Слияние

Слияние переносит изменения из двух веток в одну. Для слияния нужно перейти на ветку style и с помощью команды git merge master совместить ветки (рисунки 16-18).

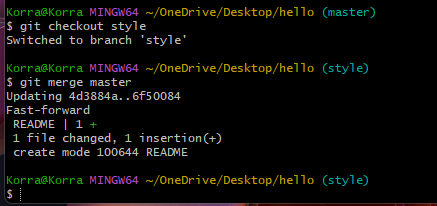


Рисунок 101 - Переключение на ветку style

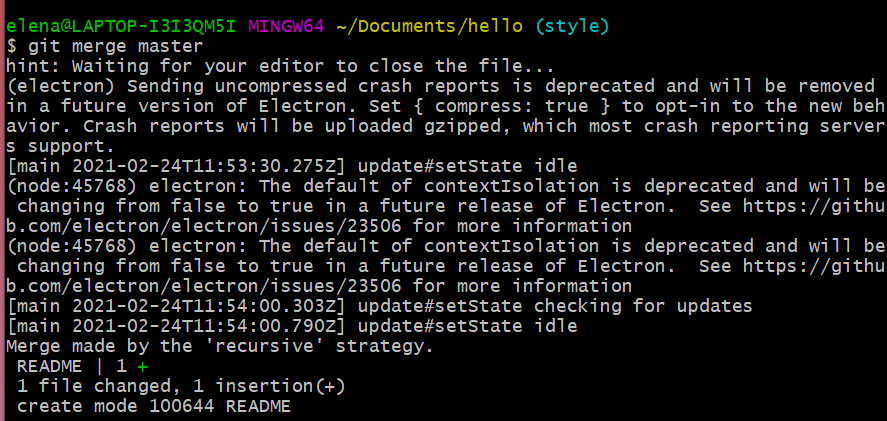


Рисунок 102 - Слияние с веткой master

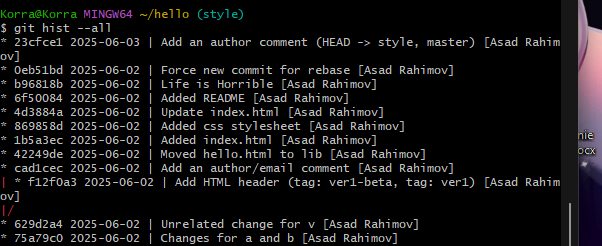
Git 

Рисунок 103 - Просмотр истории

## Создание конфликта

Для того, чтобы создать конфликт необходимо перейти в ветку master и внести изменения в файл hello.html (рисунки 19-21).

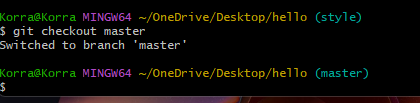


Рисунок 104 - Переход в ветку master

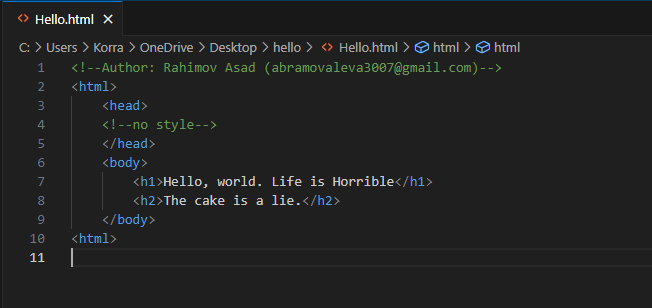


Рисунок 105 - Внесение изменений

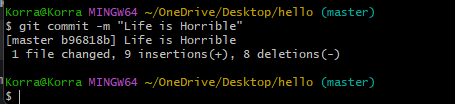


Рисунок 106 - Индексация и коммит

После выполнения предыдущих действий при просмотре веток можно будет увидеть конфликт в виде, как на рисунке 22.

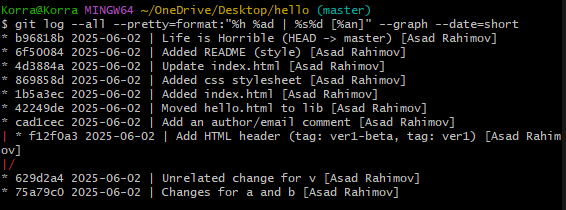


Рисунок 107 - Конфликт изменений

## Разрешение конфликтов

При попытке объединить ветку style с master будет показана ошибка из-за конфликта, как показано на рисунке 23.

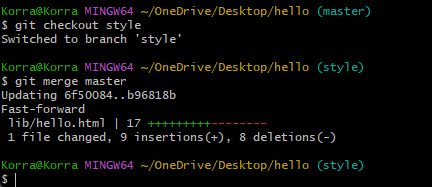


Рисунок 108 - Ошибка при слиянии

При открытии файла hello.html конфликт будет показан (рисунок 24).

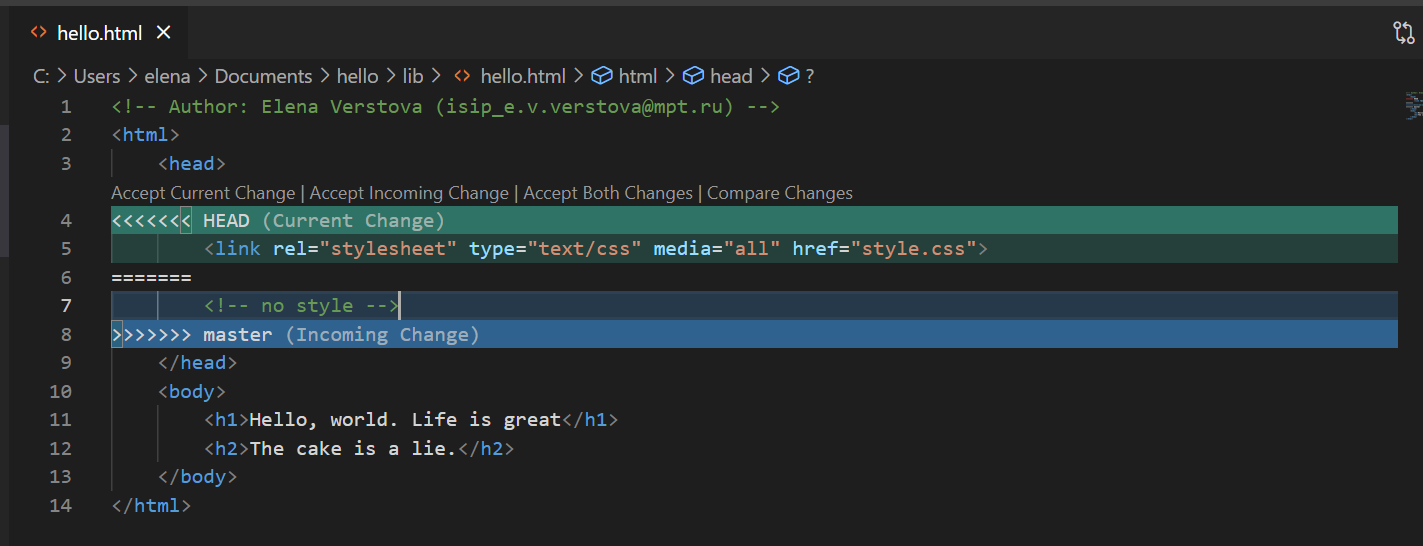


Рисунок 109 - Просмотр файла hello.html при наличии конфликта

Чтобы решить конфликт, нужно внести изменения вручную (рисунок 25).

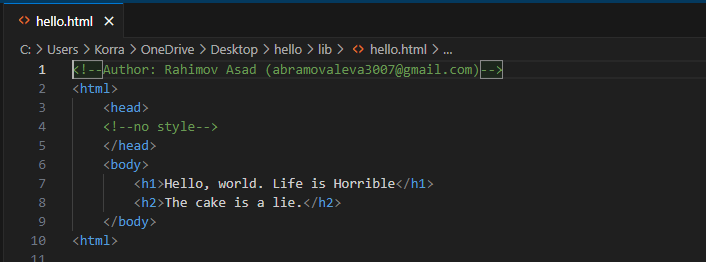


Рисунок 110 - Решение конфликта вручную

Затем следует произвести индексацию и коммит (рисунок 26).

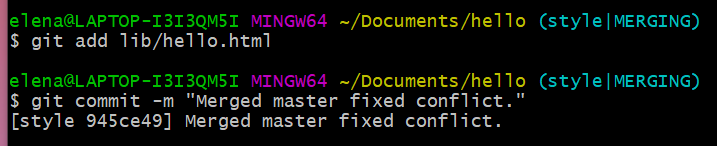


Рисунок 111 - Индексация и коммит

# \_УП\_5

# Работа с Git

## Сброс ветки style

Для сброса ветки необходимо применить команду reset --hard до требуемой точки (рисунки 1-2).

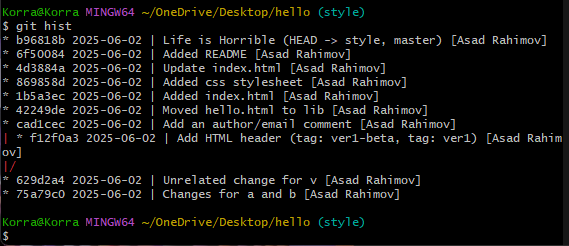
****

Рисунок 112 - Просмотр истории

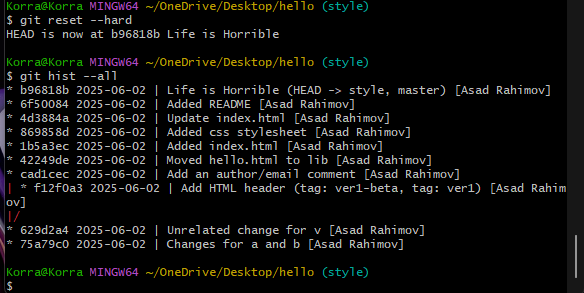
****

Рисунок 113 - Сброс ветки style

## Сброс ветки master

Аналогичные действия нужно произвести и для ветки master (рисунки 3-4).

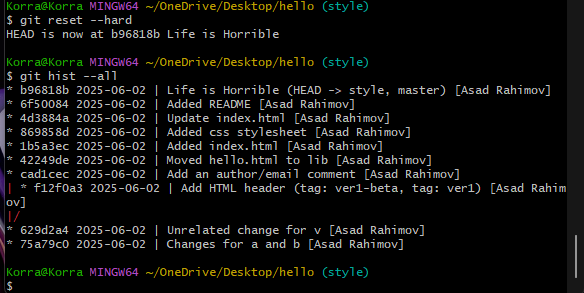
****

Рисунок 114 - Переключение на master и просмотр истории

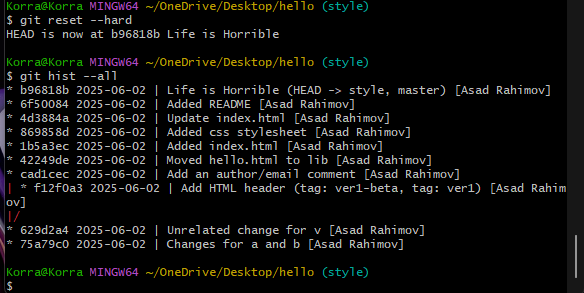
****

Рисунок 115 - Сброс ветки master

## Перебазирование

Команду rebase можно использовать вместо команды merge (рисунок 5).

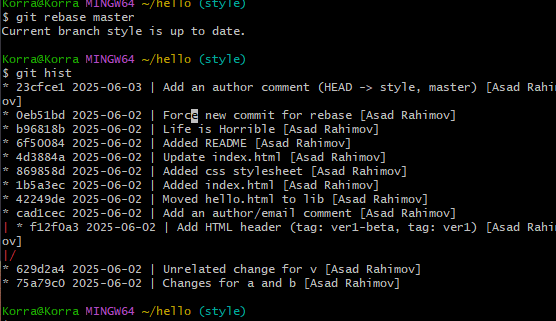


Рисунок 116 - Перебазирование веток

## Слияние в ветку master

Далее требуется произвести слияние веток с помощью merge (рисунки 6-7).

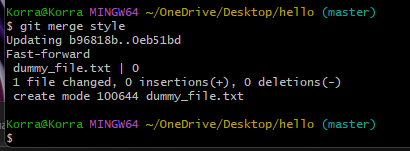
****

Рисунок 117 - Слияние веток

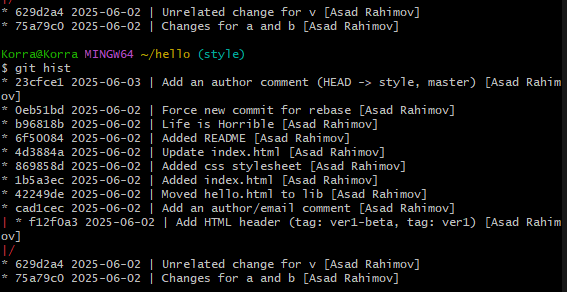
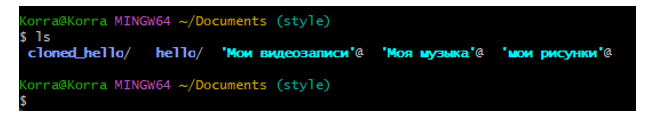


Рисунок 118 - Просмотр истории

## Клонирование репозиториев

Далее требуется научиться делать копии репозиториев. Для этого необходимо перейти в рабочий каталог и затем использовать команду git clone. Все данные действия показаны на рисунке 8.



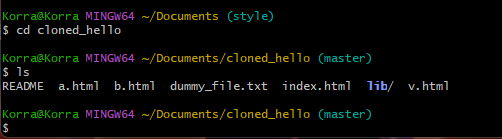
****

Рисунок 119 - Переход в рабочий каталог и его клонирование

## Просмотр клонированного репозитория

После этого можно просмотреть клонированный репозиторий (рисунки 9-10).

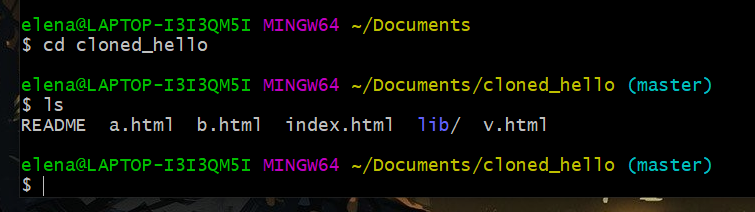


Рисунок 120 - Просмотр содержимого клонированного репозитория

****

Рисунок 121 - Просмотр логов клонированного каталога

## Origin

Origin – имя по умолчанию. Просмотр данных о нем возможен с помощью команд, показанных на рисунке 11.

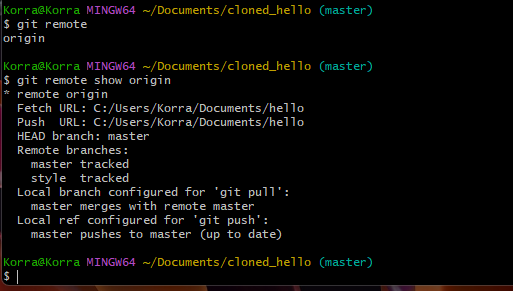
****

Рисунок 122 - Просмотр данных об origin

## Удаленные ветки

Для просмотра удаленных веток используется команда git branch -a (рисунок 12).

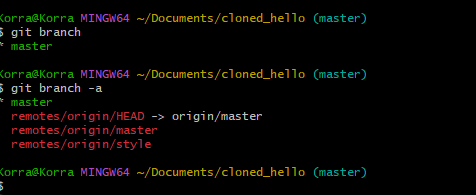
****

Рисунок 123 - Просмотр удаленных веток

# \_УП\_6

# Работа с Git

Сначала необходимо внести изменения в оригинальный репозиторий. Для этого нужно перейти в данный репозиторий (рисунок 1).

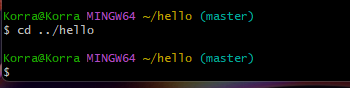


Рисунок 124 - Переход в оригинальный репозиторий

Далее надо внести изменения в файл README (рисунок 2) и затем произвести индексацию и коммит (рисунок 3).

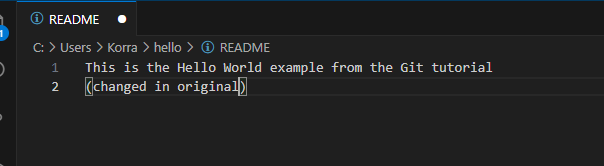


Рисунок 125 - Изменения в файле README

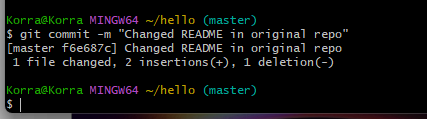


Рисунок 126 - Индексация и коммит новых изменений

Далее требуется перейти в клонированный репозиторий и извлечь изменения с помощью команды git fetch (рисунок 4) и просмотреть историю (рисунок 5).

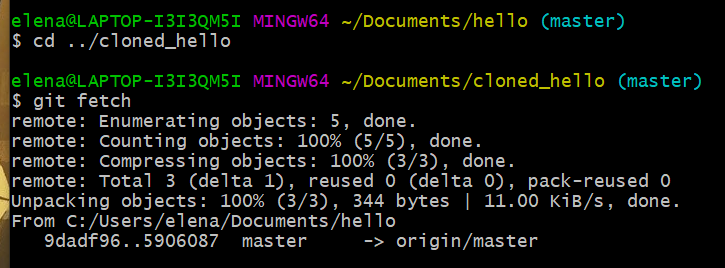


Рисунок 127 - Извлечение изменений

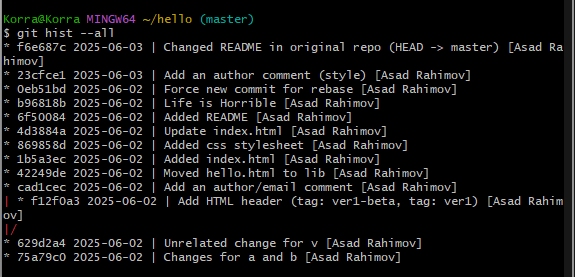


Рисунок 128 - Просмотр истории

При попытке вывести содержимое файла README можно увидеть, что изменения не были внесены (рисунок 6).



Рисунок 129 - Вывод содержимого файла README

Далее нужно слить извлеченные изменения в ветку master (рисунок 7).

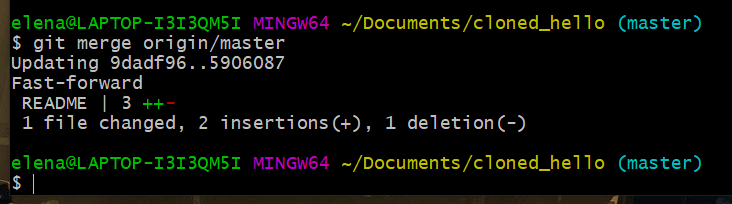


Рисунок 130 - Слияние изменений

И после выполнения предыдущего действия при выводе README можно будет увидеть последние изменения (рисунок 8).

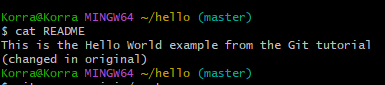


Рисунок 131 - Вывод содержимого файла README

Также существует команда, объединяющая функции git fetch и git merge, которая показана на рисунке 9.

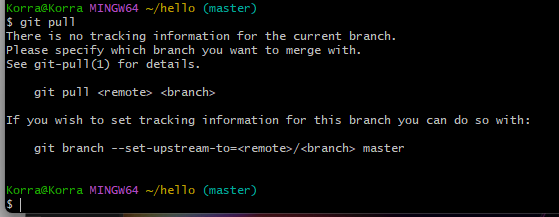


Рисунок 132 - Команда git pull

Далее требуется добавить локальную ветку, которая будет отслеживать удаленную ветку (рисунок 10).

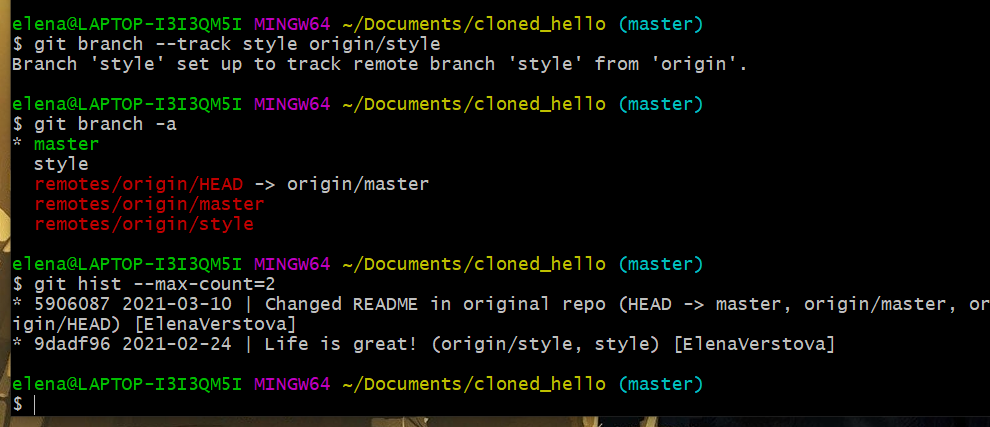


Рисунок 133 - Добавление локальной ветки

Далее необходимо создать чистый репозиторий (рисунок 11).

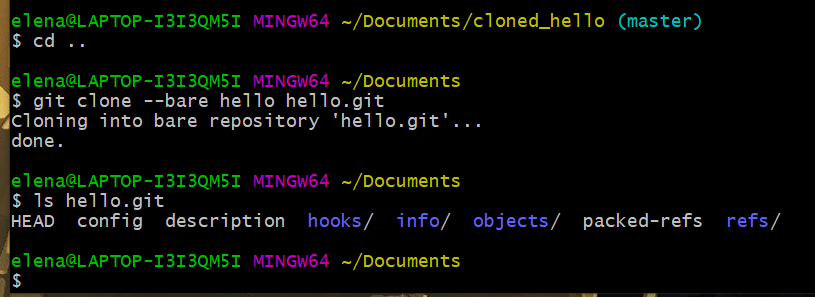


Рисунок 134 - Создание чистого репозитория

Для добавления удаленного репозитория используется команда, показанная на рисунке 12.

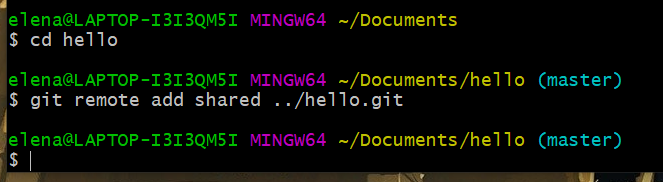


Рисунок 135 - Добавление удаленного репозитория

Затем требуется научиться отправлять изменения в удаленный репозиторий. Для этого сначала надо внести изменения, проиндексировать и произвести коммит (рисунок 13-14).

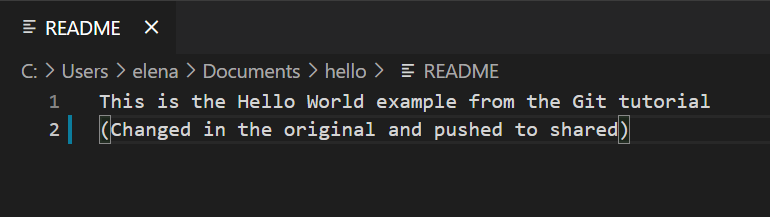


Рисунок 136 - Внесение изменений в файл

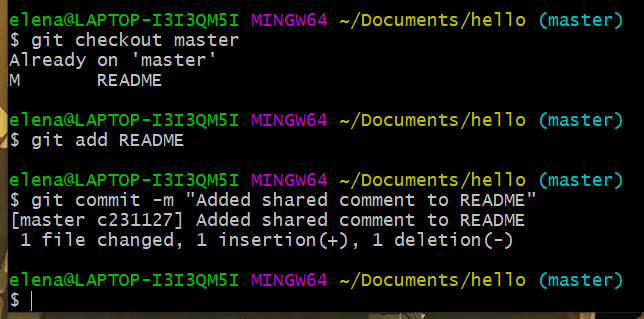


Рисунок 137 - Индексация и коммит

Далее надо отправить изменения в общий репозиторий, используя команду git push shared master (рисунок 15).

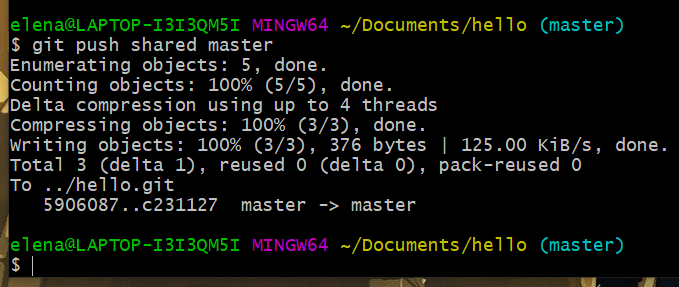


Рисунок 138 - Отправка изменений в общий репозиторий

Для извлечения общих изменений нужно перейти в клонированный каталог и выполнить перечень команд (рисунок 16).

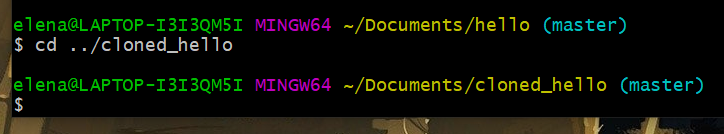


Рисунок 139 - Переход в клонированный репозиторий

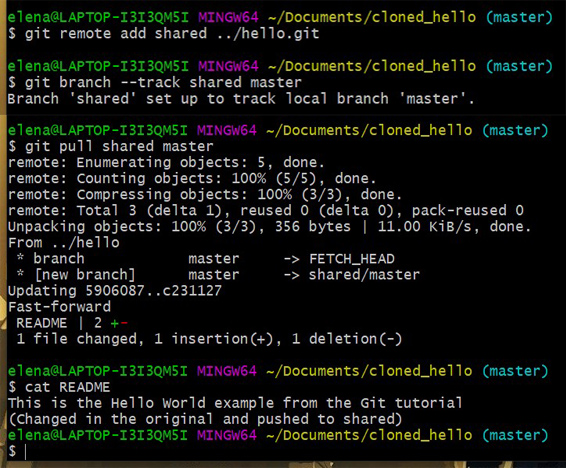


Рисунок 140 - Команды, извлекающие общие изменения

Для настройки git сервера нужно выполнить команду, показанную на рисунке 18. Затем в другом окне можно проверить работу сервера, сделав копию проекта hello (рисунок 18).

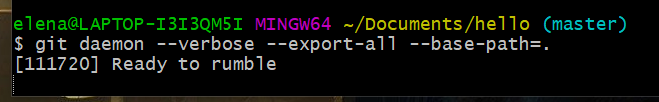


Рисунок 141 - Настройка сервера

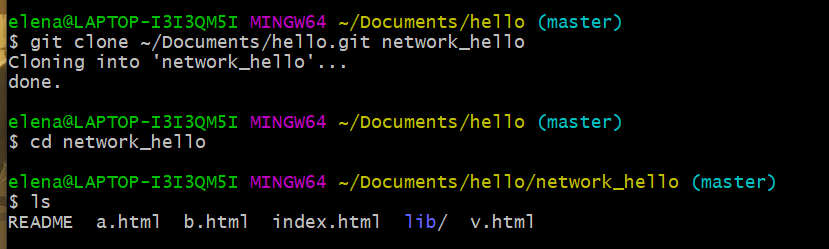


Рисунок 142 - Клонирование проекта

# \_УП\_7

# Работа с Git

Для того, чтобы просмотреть список настроенных удалённых репозиториев, необходимо запустить команду git remote (рисунок 1).

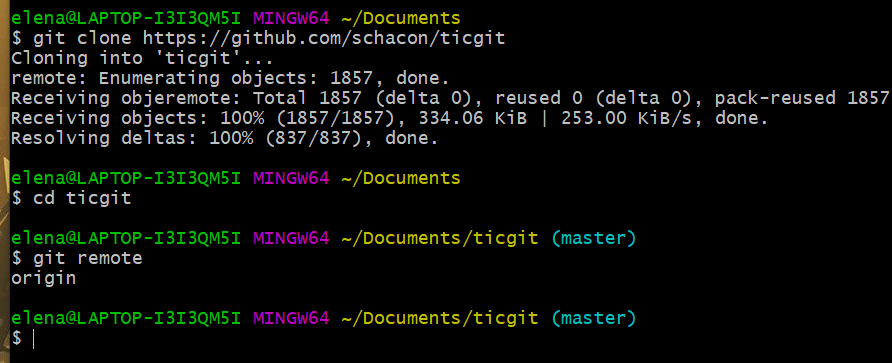


Рисунок 143 – Клонирование репозитория и просмотр удаленных репозиториев

Можно указать ключ -v, чтобы просмотреть адреса для чтения и записи, привязанные к репозиторию (рисунок 2).

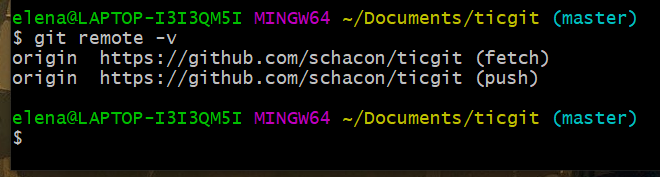


Рисунок 144 - Просмотр удаленных репозиториев с ключом -v

Для добавления удаленного репозитория с новым именем используется команда git remote add (рисунок 3).

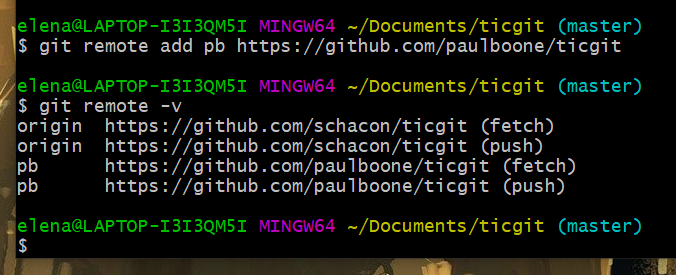


Рисунок 145 - Добавление удаленного репозитория

После задания имени репозиторию впоследствии его можно использовать вместо указания полного пути (рисунок 4).

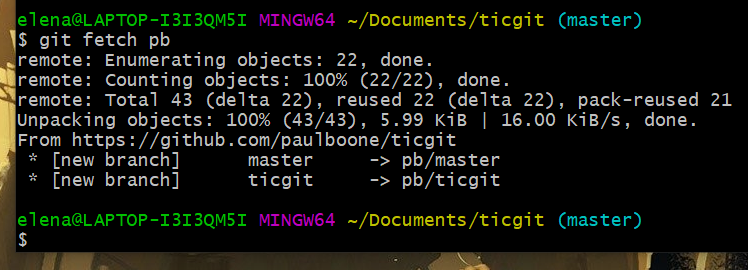


Рисунок 146 - Использование имени вместо пути

Для получения данных из удалённых проектов используется команда git fetch (рисунок 5).

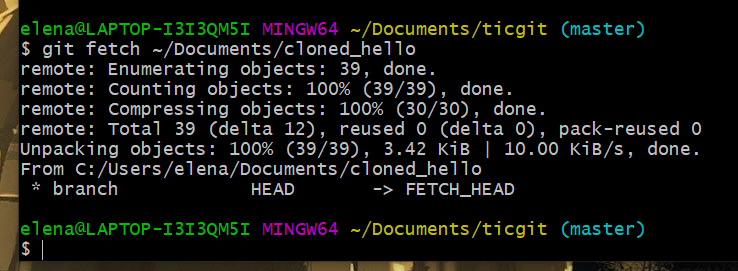


Рисунок 147 - Получение данных из удаленных проектов

Для отправки изменений в удаленный репозиторий используется команда git push (рисунок 6).

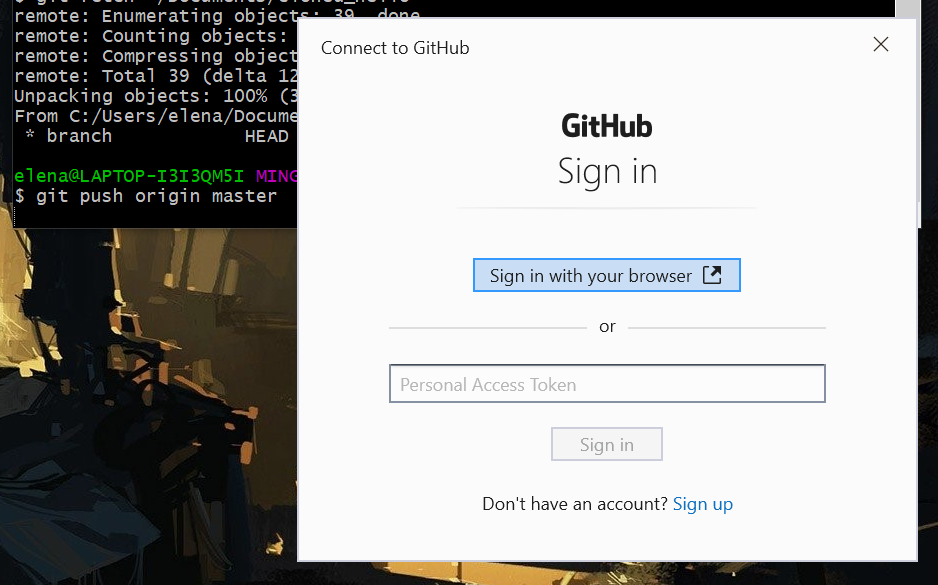


Рисунок 148 - Отправка изменений в удаленный репозиторий

Для получения информации об одном из удалённых репозиториев, можно использовать команду git remote show (рисунок 7).

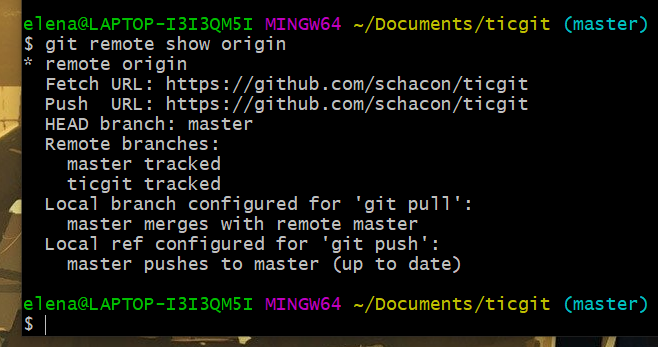


Рисунок 149 - Информация об удаленном репозитории

Для переименования удаленных репозиториев используется команда git remote rename (рисунок 8).

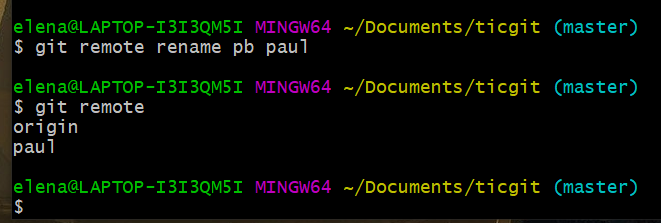


Рисунок 150 - Переименование удаленного репозитория

Для удаления удаленного репозитория нужно выполнить команду git remote remove (рисунок 9).

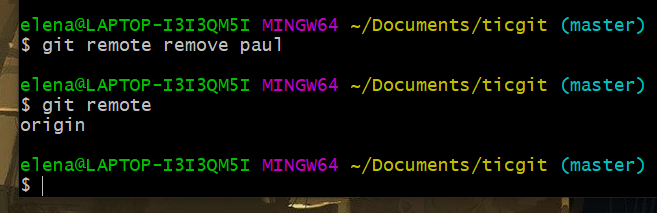


Рисунок 151 - Удаление удаленного репозитория

Просмотреть существующие теги можно с помощью команды git tag (рисунок 10).

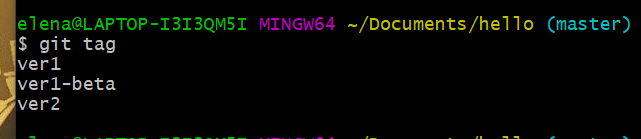


Рисунок 152 - Просмотр тегов

Для создания аннотированной метки нужно выполнить команду, показанную на рисунке 11.

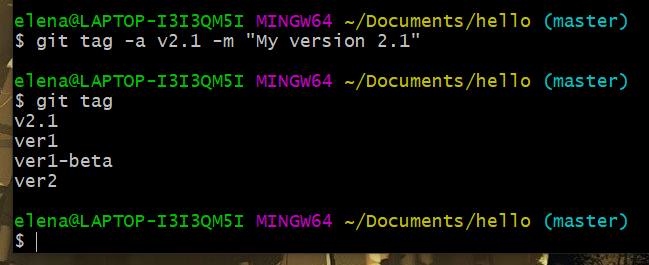


Рисунок 153 - Создание аннотированной метки

Команда git show осуществляет просмотр данных тегов вместе с коммитом (рисунок 12).

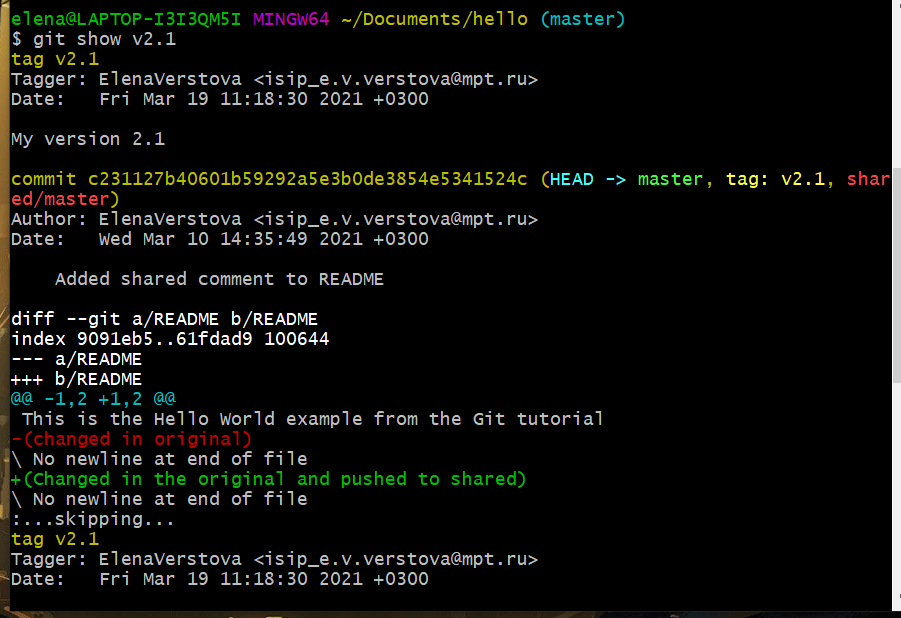


Рисунок 154 - Просмотр данных тега

Для создания легковесной метки не нужно передавать опции -a, -s и -m, надо указать только название (рисунок 13). Просмотр данных такой метки осуществляется также с помощью git show (рисунок 14).



Рисунок 155 - Создание легковесной метки

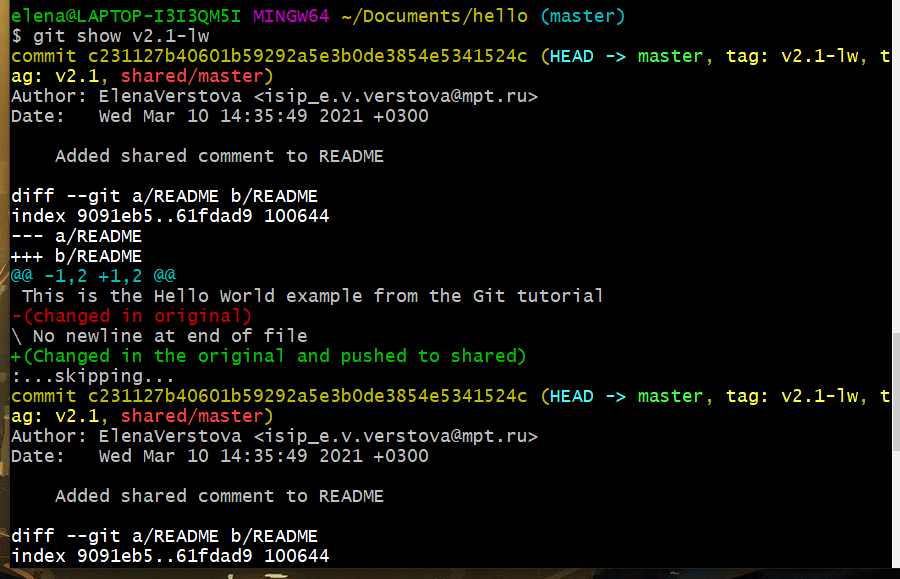


Рисунок 156 - Просмотр данных тега

Для отметки определенного коммита тегом надо указать его хэш (рисунки 15-17).

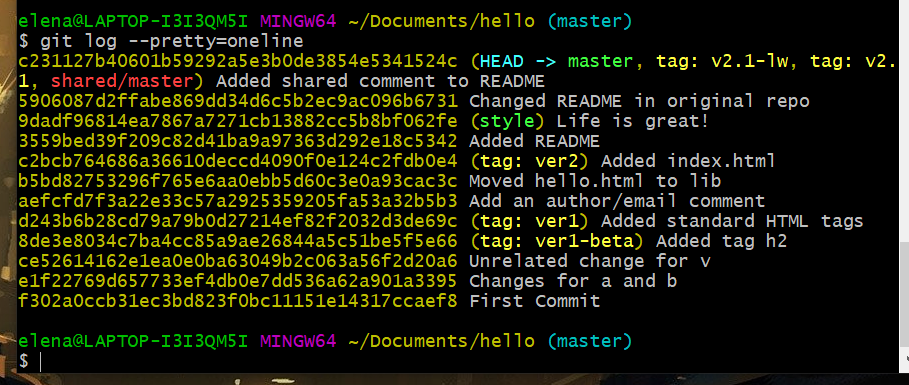


Рисунок 157 - Просмотр истории

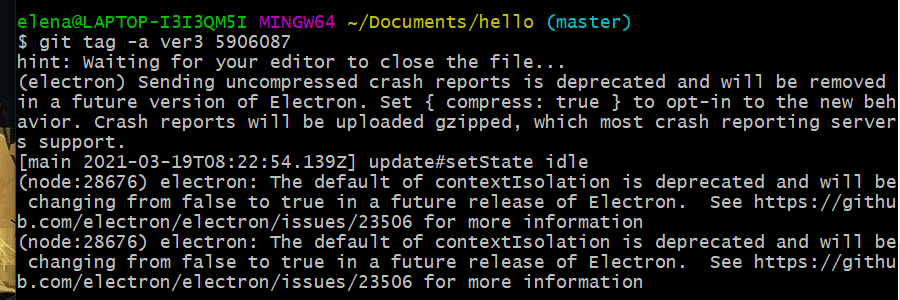


Рисунок 158 - Создание тега определенному коммиту

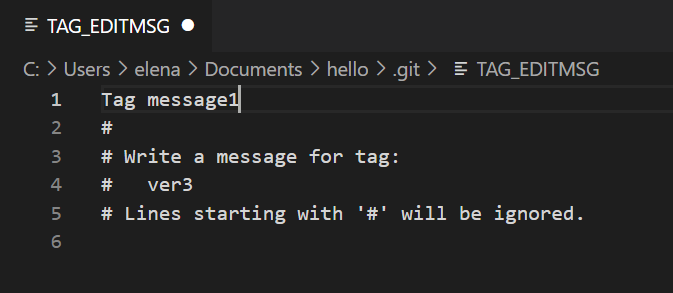


Рисунок 159 - Ввод сообщения в текстовом редакторе

Данные этого тега можно просмотреть аналогичным образом (рисунок 18).

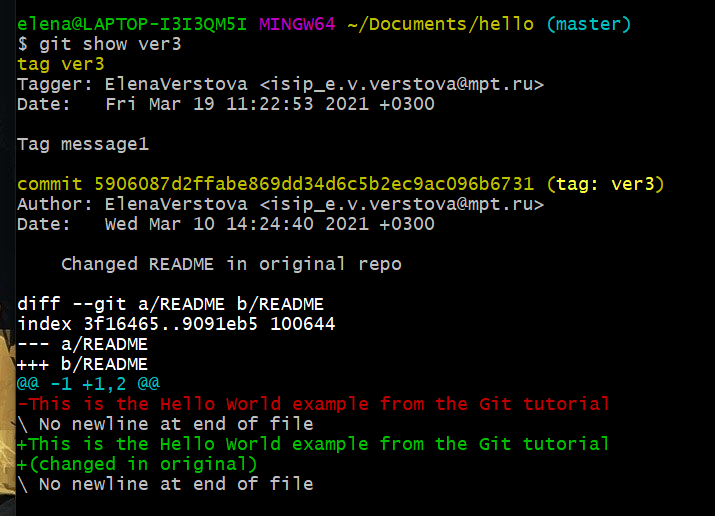


Рисунок 160 - Просмотр данных тега

По умолчанию, команда git push не отправляет теги на удалённые сервера. Нужно выполнить команду git push shared (рисунок 19).

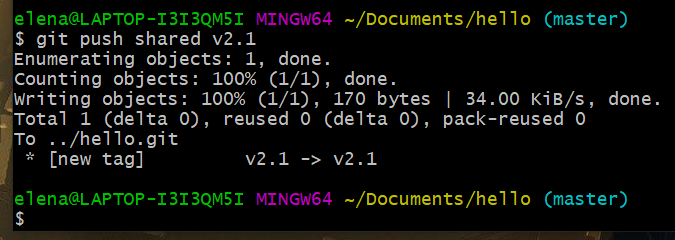


Рисунок 161 - Отправка тега на удаленный сервер

Можно использовать опцию --tags для команды git push. В таком случае все теги отправятся на удалённый сервер (рисунок 20).

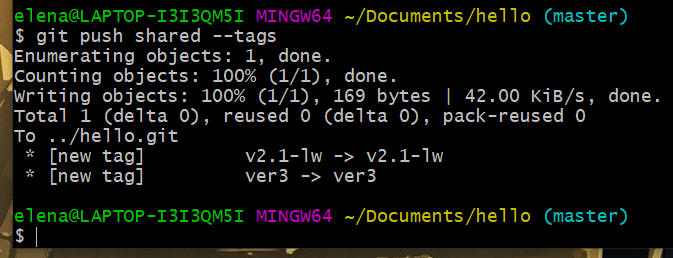


Рисунок 162 - Отправка всех тегов на сервер

Для того, чтобы удалить тег, надо использовать команду git tag с параметром -d (рисунок 21).



Рисунок 163 - Удаление тега

Для удаления тега с сервера используется команда, показанная на рисунке 22.

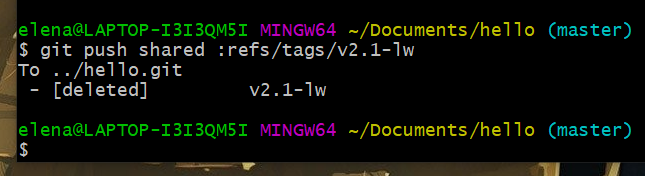


Рисунок 164 - Удаление тегов с сервера

Для того, чтобы получить версии файлов, на которые указывает тег, можно выполнить git checkout для тега. Однако, это переведёт репозиторий в состояние «detached HEAD». Если в состоянии «detached HEAD» внести изменения и сделать коммит, то тег не изменится, при этом новый коммит не будет относиться ни к какой из веток, а доступ к нему можно будет получить только по его хэшу. Поэтому в таком случае следует создать новую ветку (рисунки 23-24).

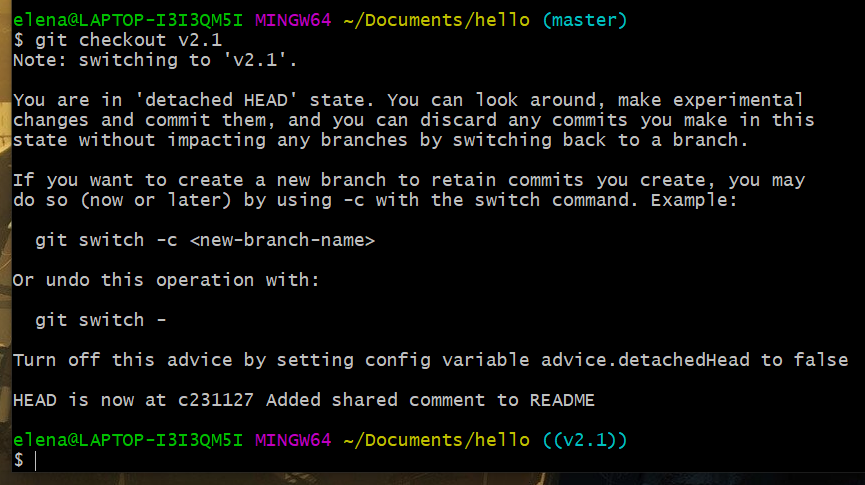


Рисунок 165 - Переключение на метку



Рисунок 166 - Создание новой ветки

Можно создать псевдонимы (алиасы) для команд. Создание алиасов и примеры их использования показаны на рисунках 25-30.

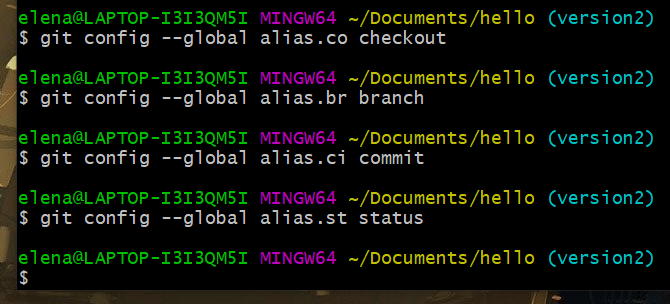


Рисунок 167 - Задание алиасов

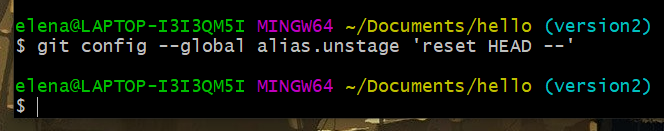


Рисунок 168 - Создание псевдонима исключения файла из индекса

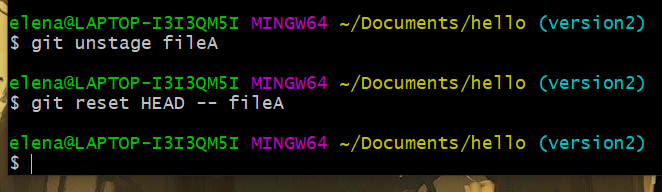


Рисунок 169 - Использование созданного псевдонима

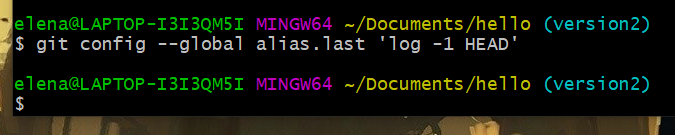


Рисунок 170 - Создание алиаса для просмотра последнего коммита

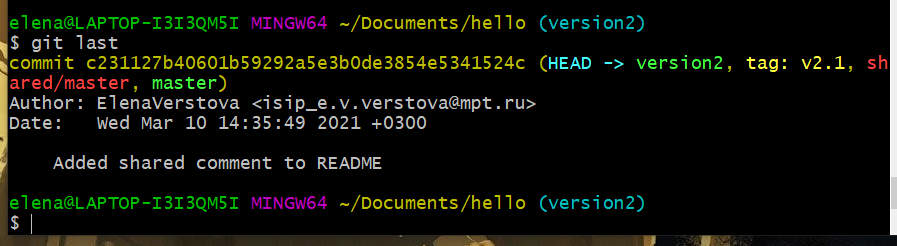


Рисунок 171 - Результат работы созданного алиаса

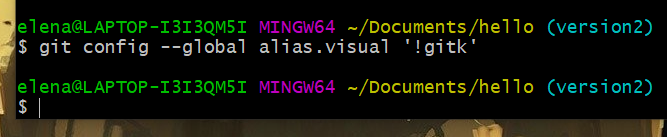


Рисунок 172 - Создание псевдонима внешней команды

# \_УП\_8

# Работа с Git

Для начала следует создать репозиторий, создать 3 файла и добавить их в коммит (рисунок 1).

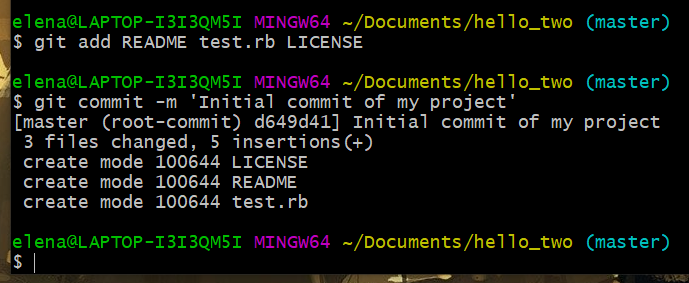


Рисунок 1 - Индексация и коммит 3 файлов

Затем надо создать ветку testing и переключиться на нее (рисунки 2-3).

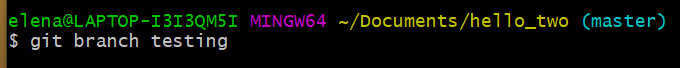


Рисунок 2 - Создание ветки testing

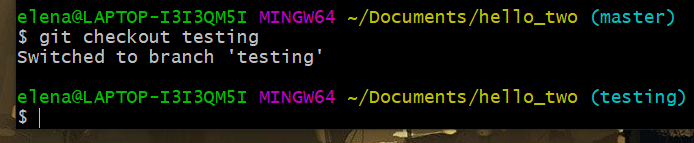


Рисунок 3 - Переключение на ветку testing

Далее надо внести изменения в файл test.rb и создать коммит (рисунок 4).

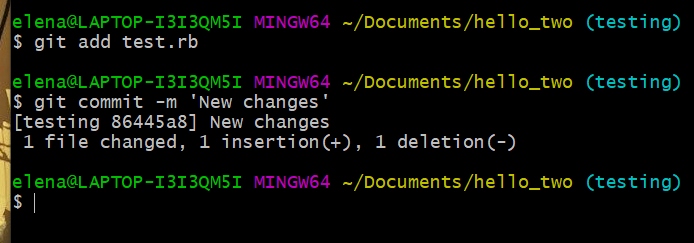


Рисунок 4 - Индексация и коммит файла test.rb

Затем необходимо переключиться на ветку master и внести изменения в файл test.rb на этой ветке (рисунки 5-6).



Рисунок 5 - Переключение на ветку master

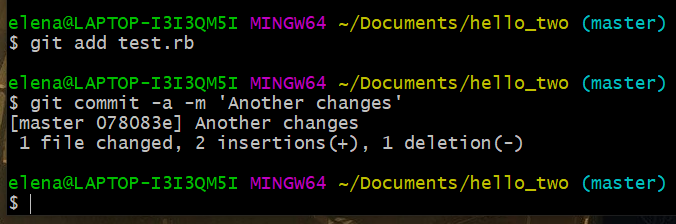


Рисунок 6 - Еще индексация и коммит test.rb

Команда git checkout -b позволяет сразу создать и переключиться на ветку (рисунок 7).

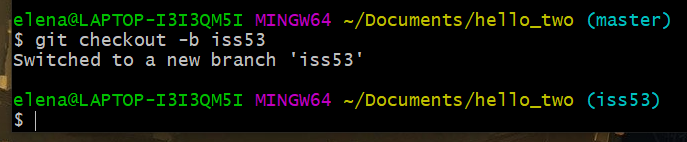


Рисунок 7 - Создание и переключение на ветку iss53

В новой ветке нужно внести в файл изменения (рисунок 8).

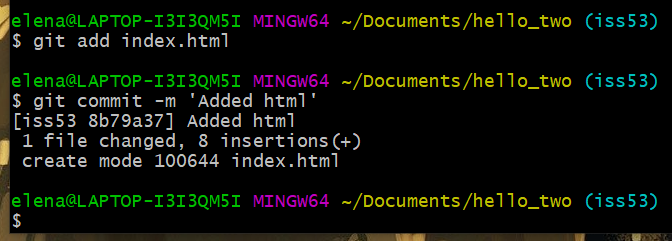


Рисунок 8 - Индексация и коммит файла index.html

Далее нужно переключить ветку на master (рисунок 9).

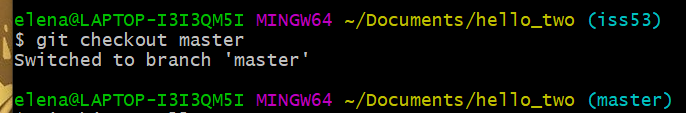


Рисунок 9 - Переключение на ветку master

Затем надо на ветке hotfix добавить изменения в файл index.html, а затем слить эту ветку и master (рисунки 10-11).

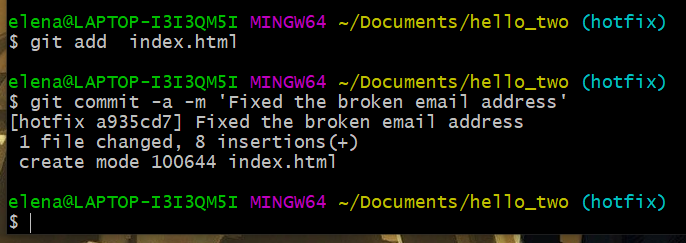


Рисунок 10 - Индексация и коммит index.html на ветке hotfix

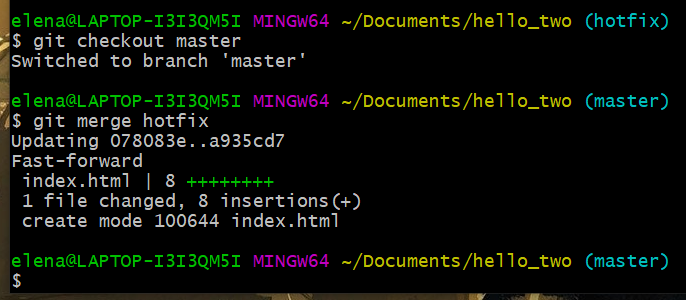


Рисунок 11 - Переключение на ветку master и объединение с веткой hotfix

После слияния ветку hotfix можно удалить (рисунок 12).1

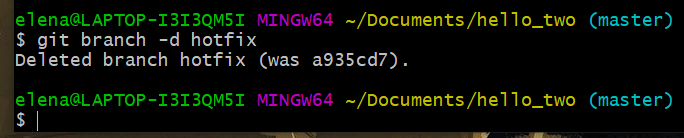


Рисунок 12 - Удаление ветки hotfix

Затем требуется внести изменения в iss53, переключиться на master и слить эти ветки (рисунки 13-14).

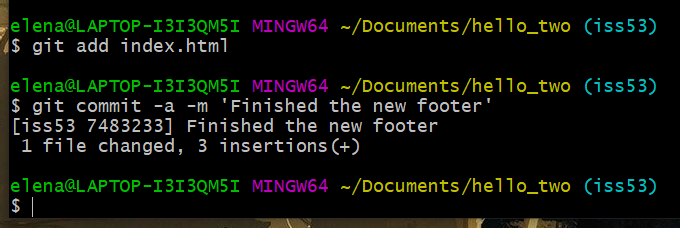


Рисунок 13 - Индексация и коммит index.html на ветке iss53

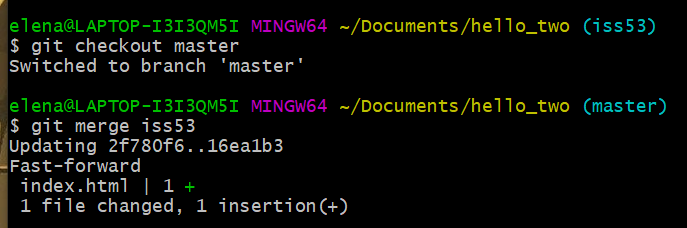


Рисунок 14 - Слияние веток

После этого ветку iss53 нужно удалить (рисунок 15).

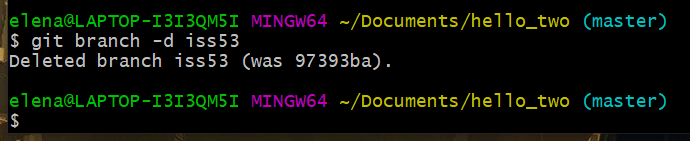


Рисунок 15 - Удаление ветки iss53

# \_УП\_9

# Работа с Git

Команда git branch делает несколько больше, чем просто создаёт и удаляет ветки. При запуске без параметров, можно получить простой список имеющихся веток (рисунок 1). Символ \*, стоящий перед веткой master указывает на ветку, на которую указывает HEAD).

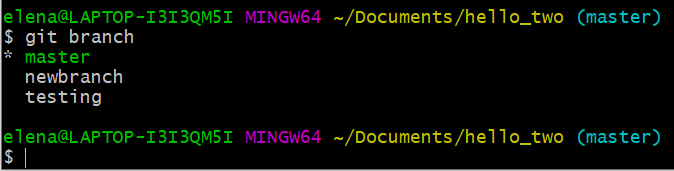


Рисунок 173 - Список существующих веток

Чтобы посмотреть последний коммит на каждой из веток, необходимо выполнить команду git branch -v (рисунок 2).

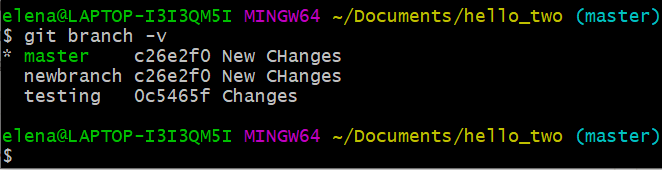


Рисунок 174 - Список веток с последними коммитами

Опции --merged и --no-merged могут отфильтровать этот список для вывода только тех веток, которые слиты или ещё не слиты в текущую ветку (рисунки 3-4).

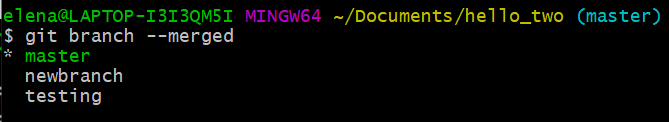


Рисунок 175 - Список веток слитых с текущей



Рисунок 176 - Список веток не слитых с текущей

Затем следует удалить ветку testing (рисунок 5). При наличии ошибок для удаления можно использовать параметр -D.

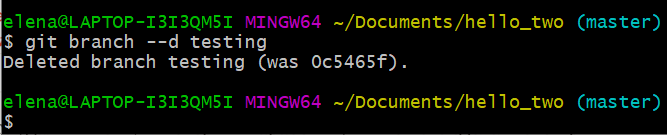


Рисунок 177 - Удаление ветки

Для получения списка удалённых веток и дополнительной информации используется команда git remote show (рисунок 6).

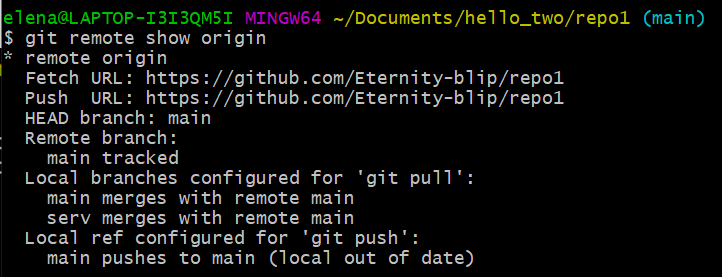


Рисунок 178 - Просмотр удаленных веток

Для отправления изменений на удалённый сервер используется команда git push <remote> <branch> (рисунок 7).

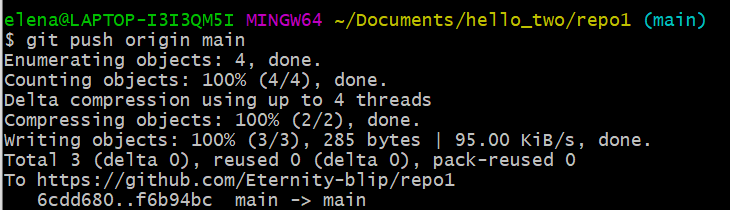


Рисунок 179 - Отправка изменений

Далее при получении обновлений с сервера будет показана ссылка на удаленную ветку (рисунок 8).

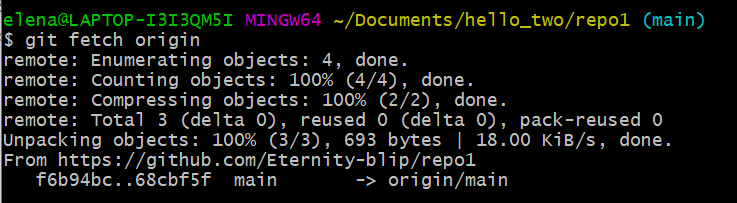


Рисунок 180 - Выполнение команды git fetch

При необходимости можно создать локальную ветку на основе удаленной (рисунок 9).

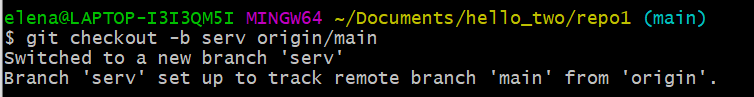


Рисунок 181 - Создание ветки на основе удаленной ветки

Для удаления веток на удаленном сервере используется команда, показанная на рисунке 10.

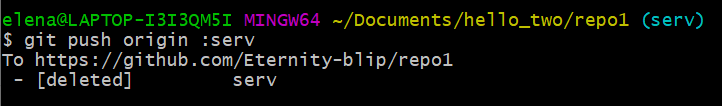


Рисунок 182 - Удаление ветки на сервере

Простой способ выполнить слияние двух веток – это команда merge. Другой способ – использование команды rebase, что означает перебазирование (рисунок 11). Это работает следующим образом: берётся общий родительский снимок двух веток (текущей, и той, поверх которой вы выполняете перебазирование), определяется дельта каждого коммита текущей ветки и сохраняется во временный файл, текущая ветка устанавливается на последний коммит ветки, поверх которой выполняется перебазирование, а затем по очереди применяются дельты из временных файлов.

Далее после этого надо переключиться на ветку master и выполнить перемотку.

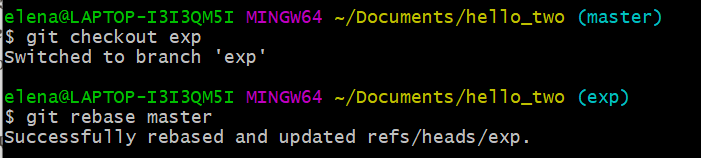


Рисунок 183 - Перемещение изменений

При наличии ответвления от ветки (сначала было ответвление на ветку se, а затем от нее на ветку cl), чтобы переместить изменения можно осуществить действия, показанные на рисунках 12-15.

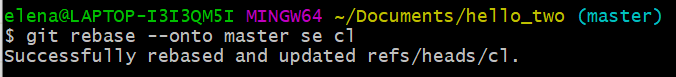


Рисунок 184 - Перемещение изменений с параметром onto

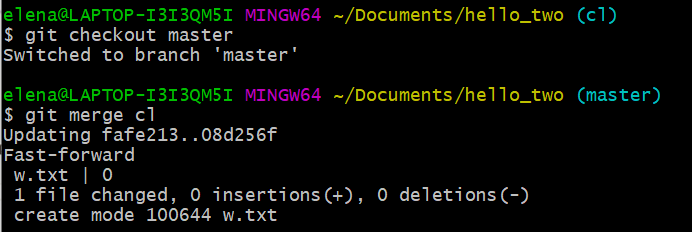


Рисунок 185 - Слияние веток master и cl

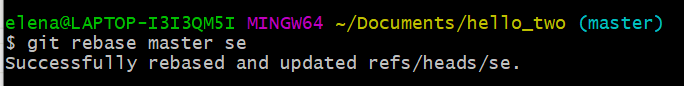


Рисунок 186 - Перемещение изменений

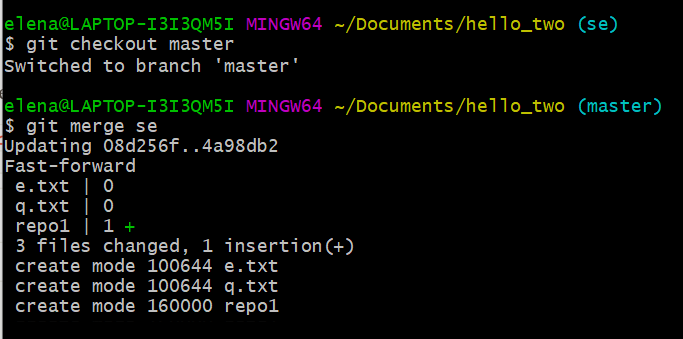


Рисунок 187 - Слияние веток master и se

После этого перемещение будет осуществлено и ветки можно удалить (рисунок 16).

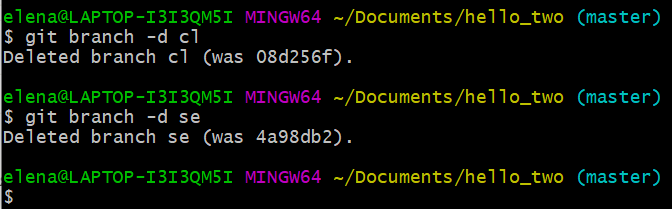


Рисунок 188 - Удаление веток cl и se