Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования

Кафедра инженерной психологии и эргономики

Дисциплина: Интерфейсы информационных систем

**Отчет**

по практической работе

на тему

**Изучение системы контроля версий Git**

Выполнил: студент группы 110101

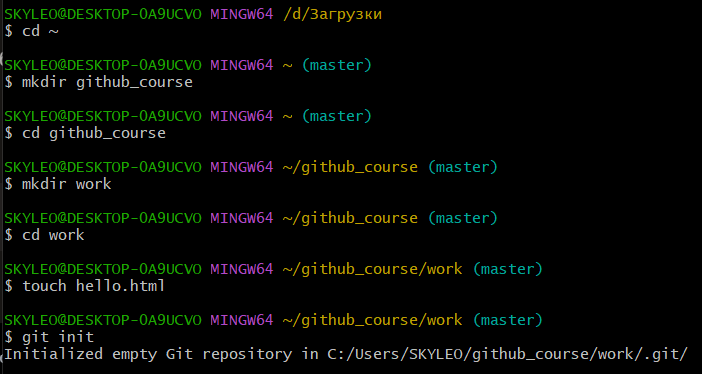
Русецкий Л. С.

Проверил: Давыдович К. И.

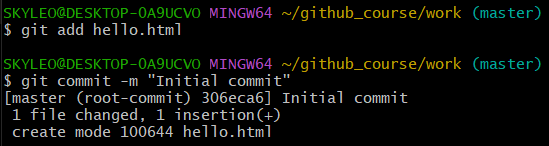
Минск 2024

**2. Создание проекта**

01 Создайте страницу «Hello, World» и репозиторий.

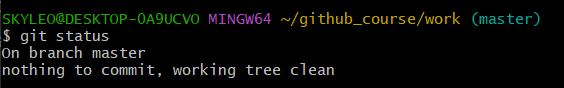


02 Добавьте страницу в репозиторий



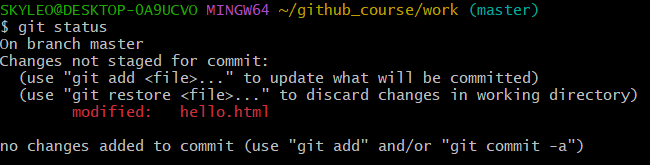
**3. Проверка состояния**

01 Проверьте состояние репозитория



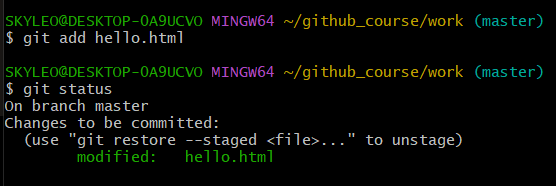
**4. Внесение изменений**

01 Измените страницу «Hello, World», проверьте состояние



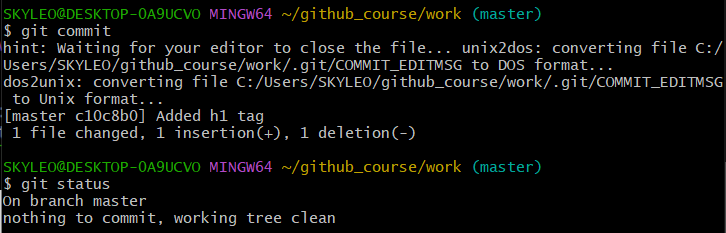
**5. Индексация изменений**

01 Добавьте изменения



**7. Коммит изменений**

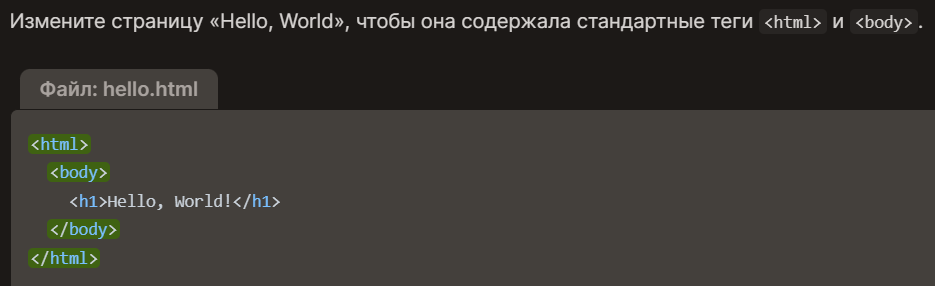
01 Закоммитьте изменения, проверьте состояние



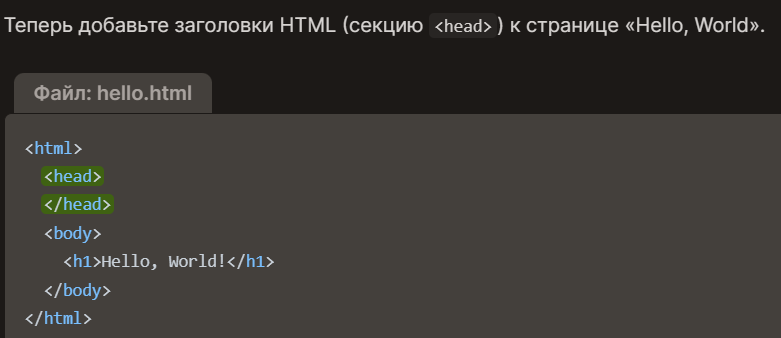
**8. Изменения, а не файлы**

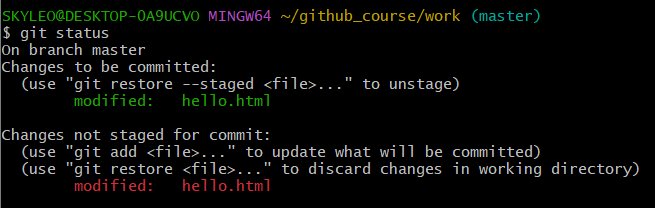
Git фокусируется на изменениях в файле, а не самом файле. Когда вы осуществляете команду git add file, вы не говорите Git добавить файл в репозиторий. Скорее вы говорите, что Git надо отметить текущее состояние файла, коммит которого будет произведен позже.

01 Первое изменение: Добавьте стандартные теги страницы, добавьте это изменение



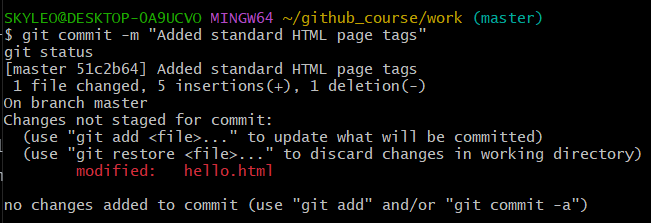
02 Второе изменение: Добавьте заголовки HTML, проверьте текущий статус





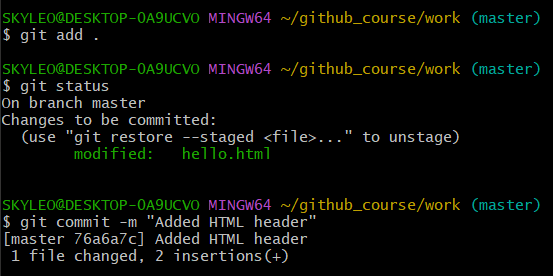
Обратите внимание на то, что hello.html указан дважды в состоянии. Первое изменение (добавление стандартных тегов) проиндексировано и готово к коммиту. Второе изменение (добавление заголовков HTML) является непроиндексированным. Если бы вы делали коммит сейчас, заголовки не были бы сохранены в репозиторий.

03 Коммит

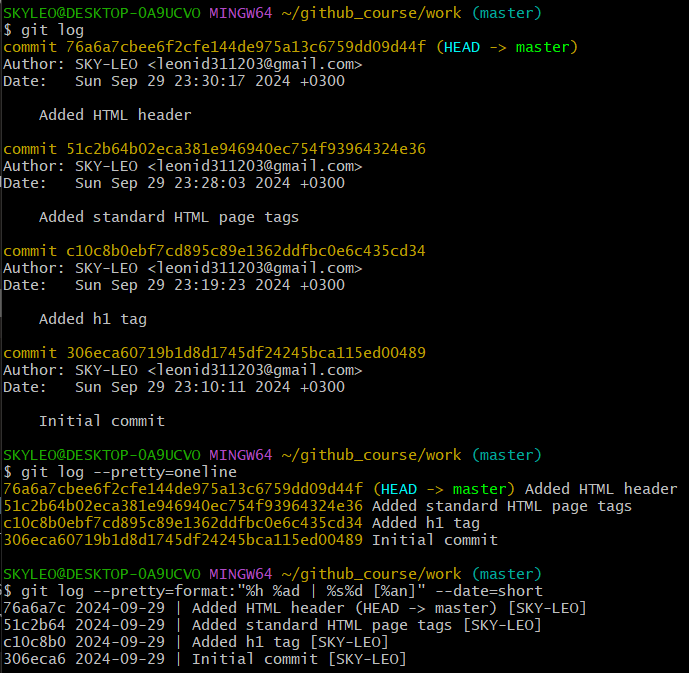


Команда status показывает, что в файле hello.html ещё есть незаписанные изменения, но область подготовки уже пуста.

04 Добавьте второе изменение, сделайте коммит второго изменения

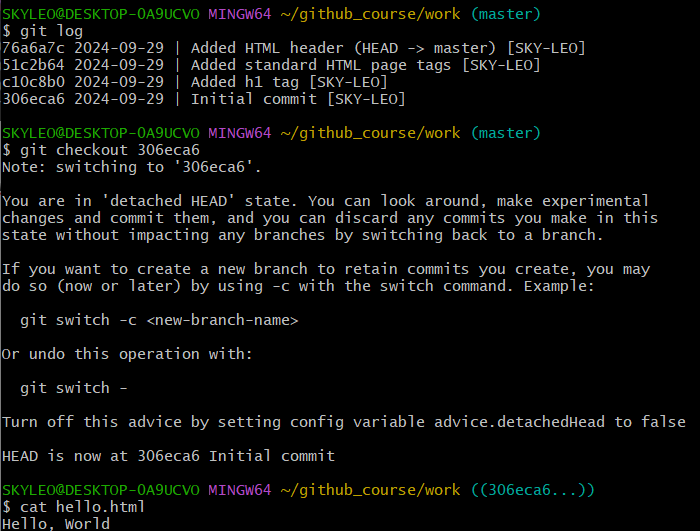


**9. История**

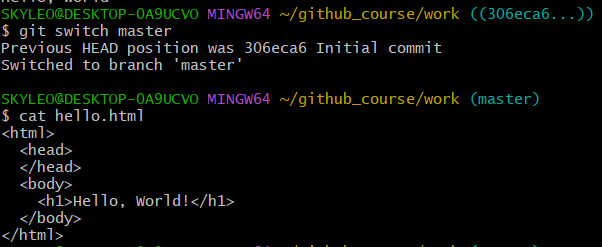


**10. Получение старых версий**

01 Получите хеши предыдущих коммитов

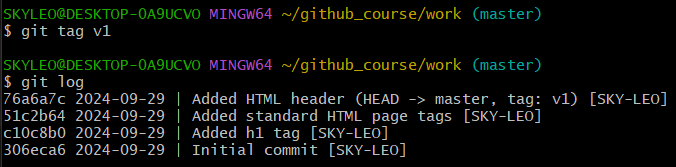


02 Вернитесь к последней версии в ветке main



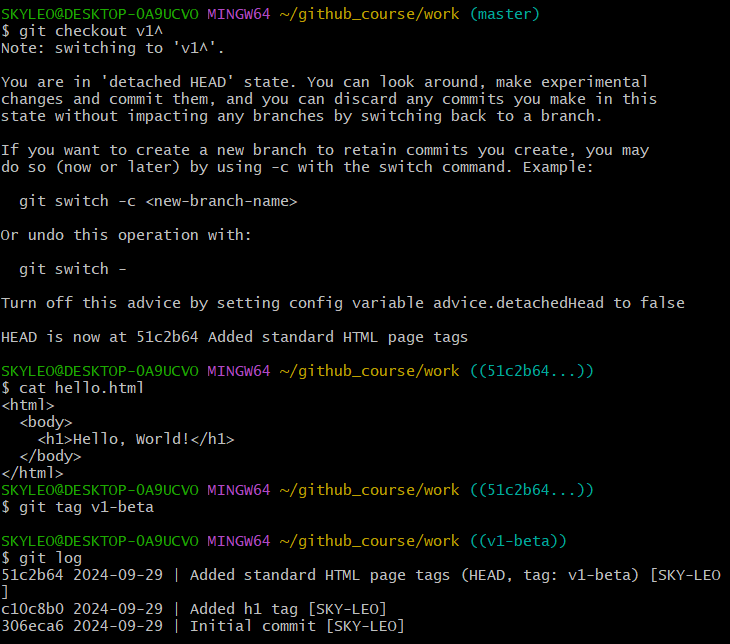
**11. Создание тегов версий**

01 Создайте тег первой версии

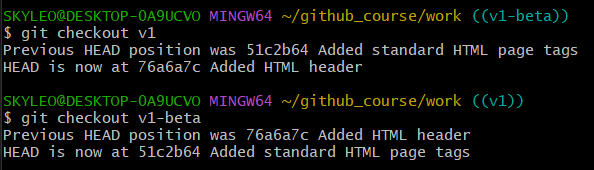


02 Теги для предыдущих версий

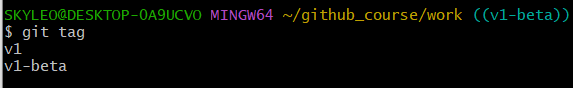
Вместо того чтобы искать хеш коммита, мы будем использовать обозначение ^, а именно v1^, указывающее на коммит, предшествующий v1.



03 Переключение по имени тега

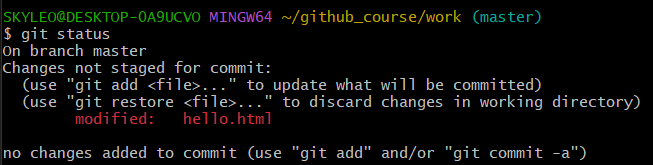


04 Просмотр тегов с помощью команды tag

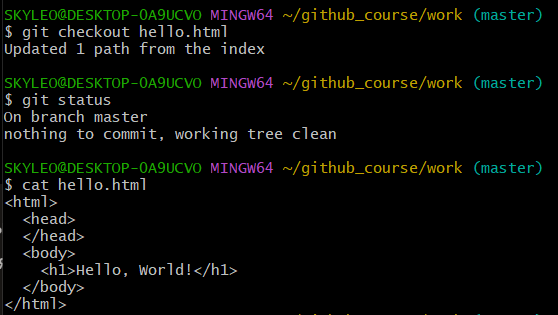


**12. Отмена локальных изменений (до индексации)**

01 Измените hello.html, Проверьте состояние

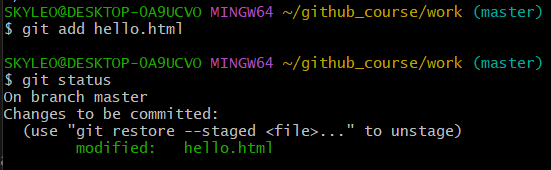


02 Отмена изменений в рабочем каталоге

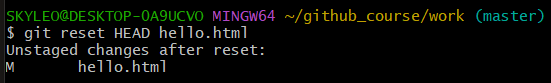


**13. Отмена проиндексированных изменений (перед коммитом)**

01 Измените файл и проиндексируйте изменения, проверьте состояние

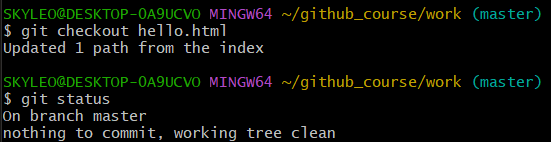


02 Сбросьте области подготовки



Команда reset (по умолчанию) не изменяет рабочую директорию. Поэтому рабочая директория всё еще содержит нежелательный комментарий. Мы можем использовать команду checkout из предыдущего урока, чтобы убрать нежелательные изменения в рабочей директории.

03 Переключитесь на версию коммита

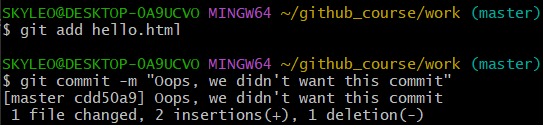


Наша рабочая директория опять чиста.

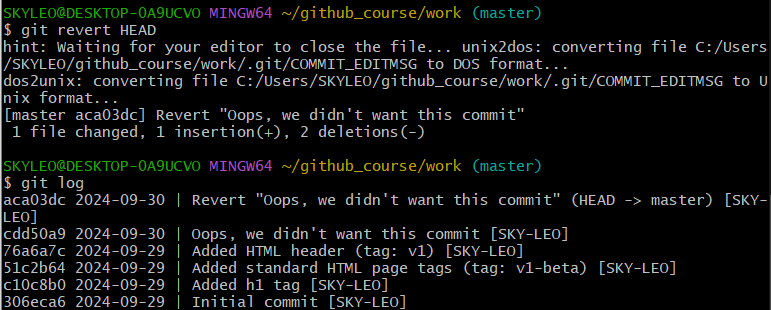
**14. Отмена коммитов**

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

01 Измените файл и сделайте коммит



02 Сделайте коммит с новыми изменениями, отменяющими предыдущие, проверьте лог

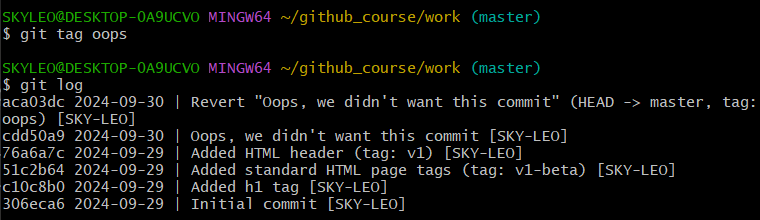


**15. Удаление коммитов из ветки**

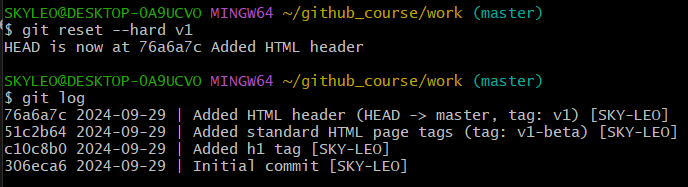
Revert из предыдущего раздела является мощной командой, которая позволяет отменить любые коммиты в репозиторий. Однако, и оригинальный и «отмененный» коммиты видны в истории ветки (при использовании команды git log).

Часто мы делаем коммит, и сразу понимаем, что это была ошибка. Было бы неплохо иметь команду «возврата», которая позволила бы нам сделать вид, что неправильного коммита никогда и не было. Команда «возврата» даже предотвратила бы появление нежелательного коммита в истории git log.

01 Проверьте нашу историю, для начала отметьте эту ветку

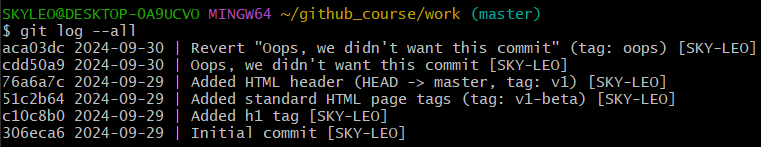


02 Сброс к коммиту, предшествующему oops



03 Ничего никогда не теряется

Что же случается с ошибочными коммитами? Оказывается, что коммиты все еще находятся в репозитории. На самом деле, мы все еще можем на них ссылаться.



Мы видим, что ошибочные коммиты не исчезли. Они все еще находятся в репозитории. Просто они отсутствуют в ветке main. Если бы мы не отметили их тегами, они по-прежнему находились бы в репозитории, но не было бы никакой возможности ссылаться на них, кроме как при помощи хешей этих коммитов. Коммиты, на которые нет ссылок, остаются в репозитории до тех пор, пока не будет запущен сборщик мусора.

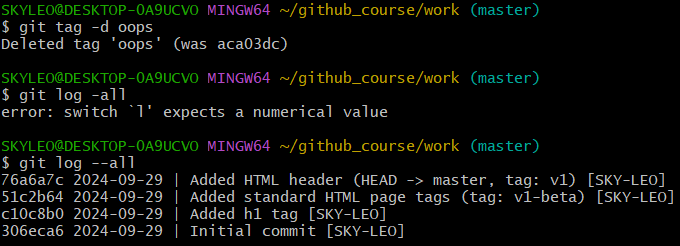
Сброс в локальных ветках, как правило, безопасен. Последствия любой «аварии» как правило, можно восстановить простым сбросом с помощью нужного коммита.

Однако, если ветка уже стала общедоступной на удаленных репозиториях, сброс может сбить с толку других пользователей ветки.

**16. Удаление тега oops**

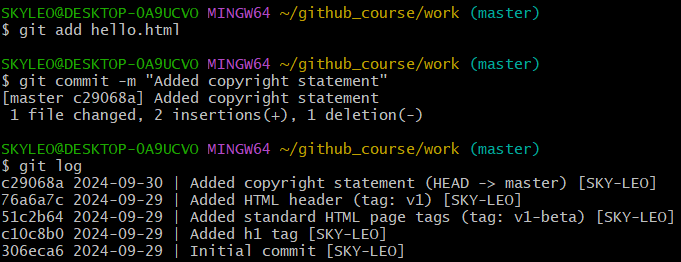
01 Удаление тега oops

Тег oops свою функцию выполнил, давайте удалим его. Это позволит внутреннему механизму Git убрать остаточные коммиты, на которые теперь не ссылаются никакие ветки или теги.

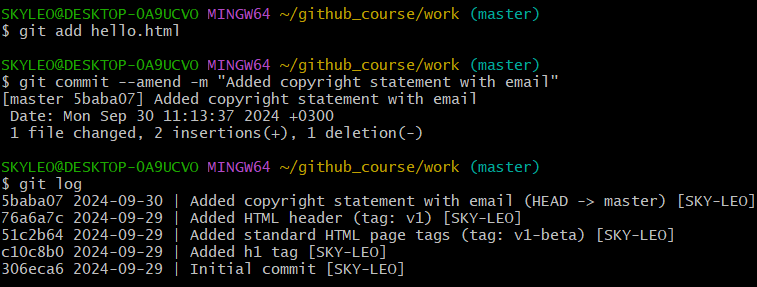


**17. Внесение изменений в коммиты**

01 Измените страницу, а затем сделайте коммит

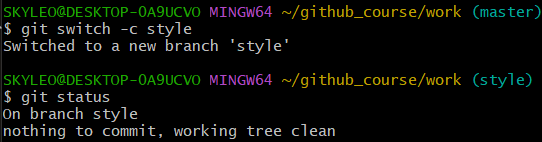


02 Ой... необходим email, измените предыдущий коммит, просмотр истории

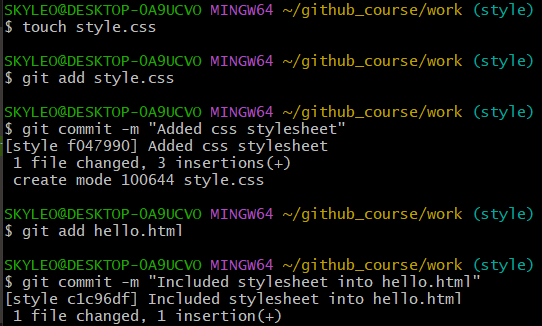


**18. Создание ветки**

01 Создайте ветку

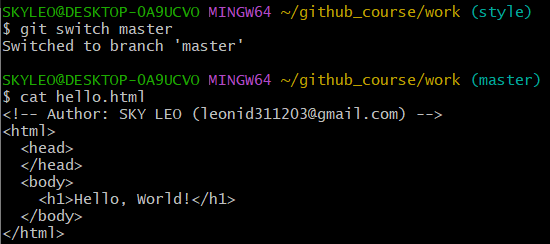


02 Добавьте файл стилей style.css, измените hello.html, чтобы он использовал style.css

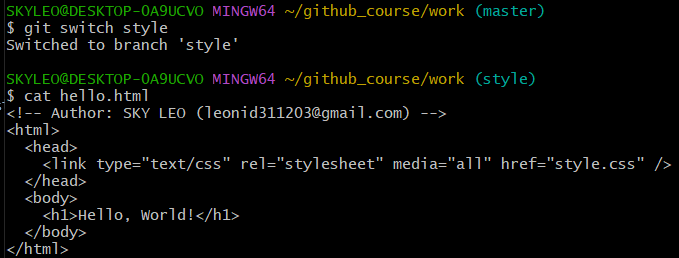


**19. Переключение веток**

01 Переключение на ветку main

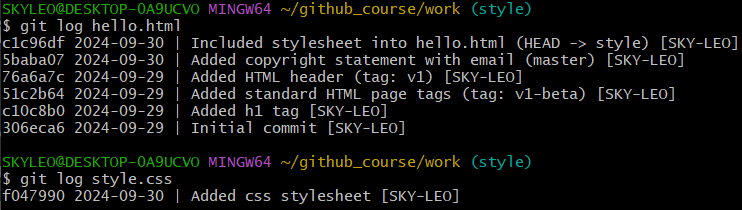


02 Вернемся к ветке style



**20. Перемещение файлов**

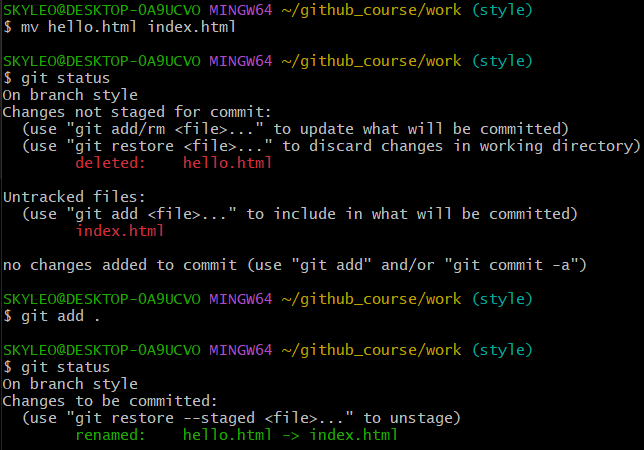
01 Просмотр истории изменений в конкретном файле



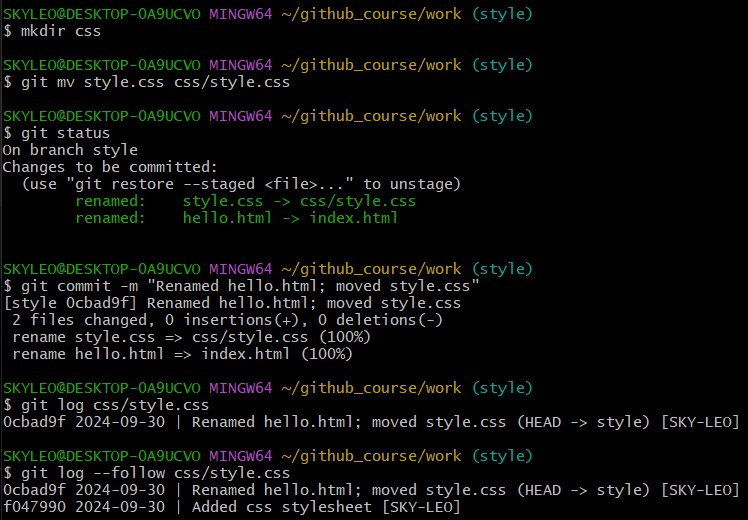
02 Просмотр различий для конкретного файла



03 Переименуйте файл hello.html

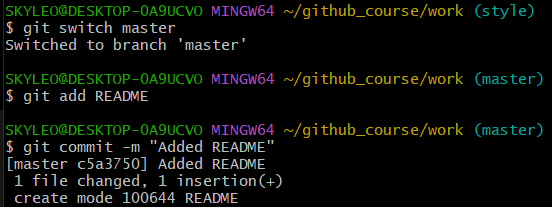


04 Переместите style.css безопасным способом



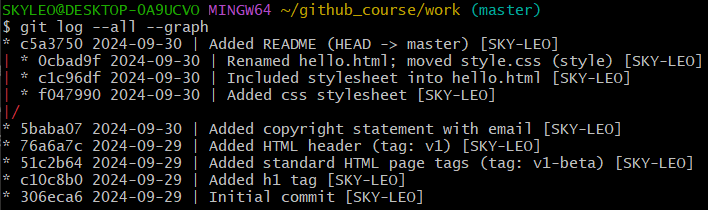
**21. Изменения в ветке main**

01 Создайте файл README, сделайте коммит файла README в ветку main



**22. Просмотр отличающихся веток**

01 Просмотрите текущие ветки

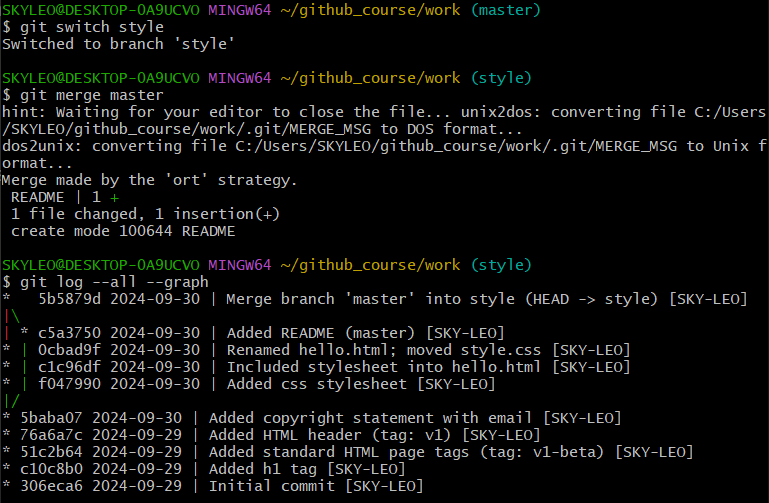


Опция --all гарантирует, что мы видим все ветки, так как по умолчанию в логе показывается только текущая ветка.

Опция --graph добавляет простое дерево коммитов, представленное в виде простых текстовых линий.

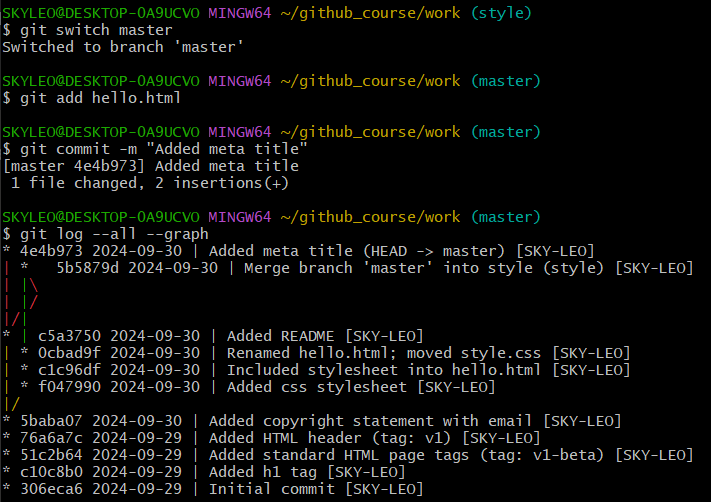
**23. Слияние**

01 Слияние веток



**24. Создание конфликта**

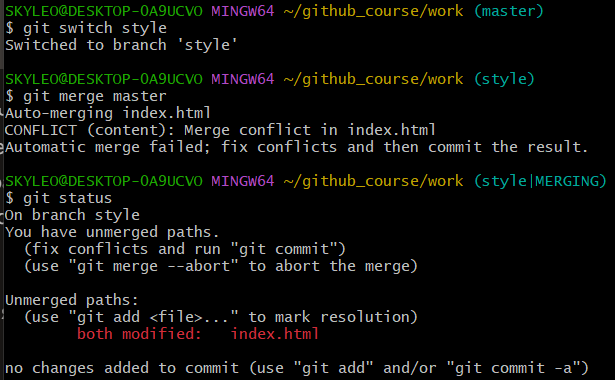
01 Вернитесь в main и создайте конфликт, просмотр веток



После коммита «Added README» ветка main была объединена с веткой style, но в настоящее время в main есть дополнительный коммит, который не был слит с style.

**25. Разрешение конфликтов**

01 Слияние main в ветку style

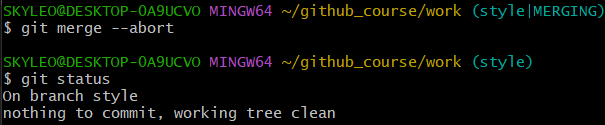




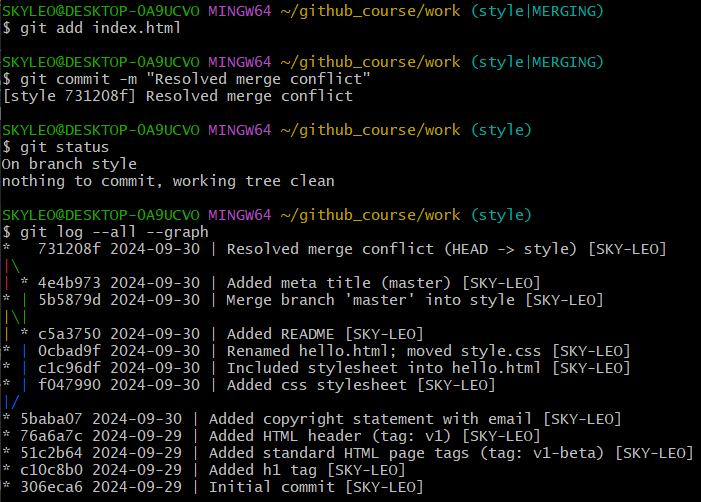
Часть между <<<<<<< >>>>>>> является конфликтом. Верхняя часть соответствует ветке style, которая является текущей веткой (или HEAD) репозитория. Нижняя часть соответствует изменениям из ветки main. Git не может решить, какие изменения применить, поэтому он просит вас разрешить конфликт вручную.

02 Отмена слияния

Прежде чем мы приступим к разрешению нашего конфликта, хочу заметить, что сразу бросаться к разрешению конфликта не всегда оптимально. Конфликт может быть вызван изменениями, о которых вы не знаете. Или же изменения слишком велики, чтобы разрешить конфликт сразу. По этой причине Git позволяет прервать слияние и вернуться к предыдущему состоянию. Для этого можно воспользоваться командой git merge --abort, как это было предложено командой status, которую мы выполнили ранее.



03 Решение конфликта, закоммитьте разрешенный конфликт



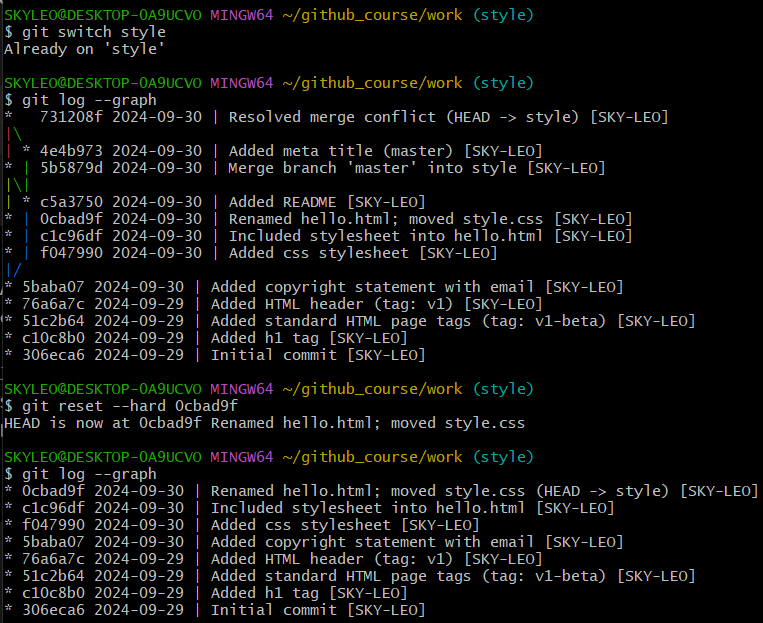
**26. rebase против merge**

Давайте рассмотрим различия между слиянием и перебазированием. Для того чтобы это сделать, нам нужно вернуться в репозиторий в момент до первого слияния, а затем повторить те же действия, но с использованием перебазирования вместо слияния.

Мы будем использовать команду reset для возврата веток к предыдущему состоянию.

**27. Сброс ветки style**

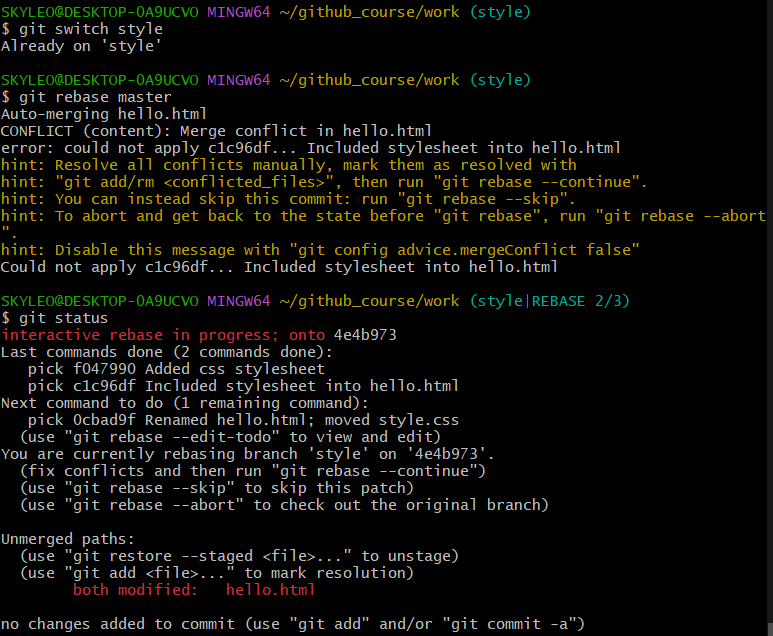
01 Сбросьте ветку style, проверьте ветку



**28. Перебазирование**

Мы вернули ветку style к точке перед первым слиянием. При этом в ветке main есть два коммита, которых нет в ветке style: новый файл README и конфликтующее изменение в файле index.html. На этот раз мы перенесем эти изменения в ветку style с помощью команды rebase, а не merge.

01 Перебазируем ветку style на ветку main



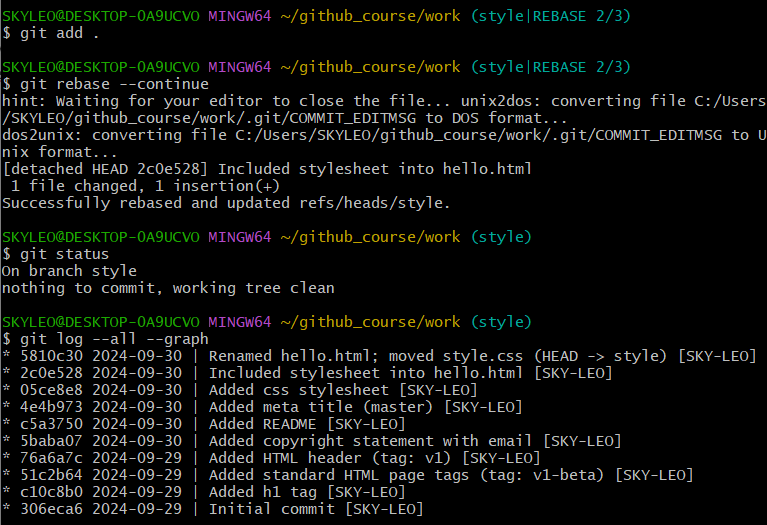
Опять возник конфликт! Обратите внимание, что конфликт возник в файле hello.html, а не в файле index.html, как в прошлый раз. Это связано с тем, что rebase находился в процессе применения изменений style поверх ветки main. Файл hello.html в main еще не был переименован, поэтому он все еще имеет старое имя.

При слиянии возник бы «обратный» конфликт. При слиянии изменения ветки main были бы применены поверх ветки style. В ветке style файл переименован, поэтому конфликт возник бы в файле index.html.

02 Разрешение конфликта



Но после этого нам не нужно коммитить изменения. Мы можем просто добавить файл в индекс и продолжить процесс rebase. Вот почему я люблю rebase! Он позволяет мне устранять конфликты, не создавая кучу уродливых конфликтов слияния.



03 Слияние VS перебазирование

Конечный результат перебазирования очень похож на результат слияния. Ветка style в настоящее время содержит все свои изменения, а также все изменения ветки main. Однако, дерево коммитов значительно отличается. Дерево коммитов ветки style было переписано таким образом, что ветка main является частью истории коммитов. Это делает цепь коммитов линейной и гораздо более читабельной.

04 Когда использовать команду rebase, а когда команду merge?

Используйте команду rebase:

* Когда вы получаете изменения из удаленного репозитория и хотите применить их к своей локальной ветке;
* Если вы хотите, чтобы история коммитов была линейной и легко читаемой.

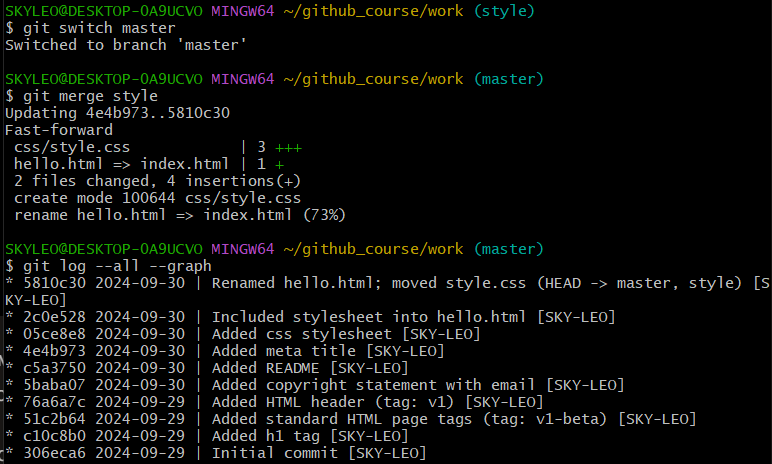
Не используйте команду rebase:

* Если текущая ветка является публичной и общей. Переписывание таких веток будет мешать работе других членов команды;
* Если важна точная история ветки коммитов (поскольку команда rebase переписывает историю коммитов).

Учитывая приведенные выше рекомендации, я предпочитаю использовать команду rebase для краткосрочных, локальных веток и команду merge для веток в публичном репозитории.

**29. Слияние в ветку main**

01 Слейте style в main, просмотрите логи

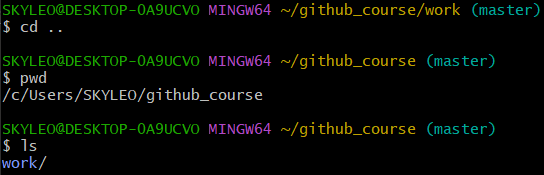


Поскольку последний коммит в main предшествует последнему коммиту ветки style, Git может выполнить ускоренное слияние, просто переместив указатель ветки вперед, на тот же коммит, что и ветка style.

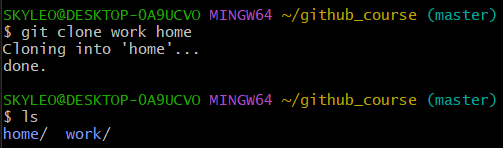
При ускоренном слиянии конфликты не возникают. Кроме того, при ускоренном слиянии не создается фиксация слияния.

**30. Клонирование репозиториев**

01 Перейдите в директорию repositories

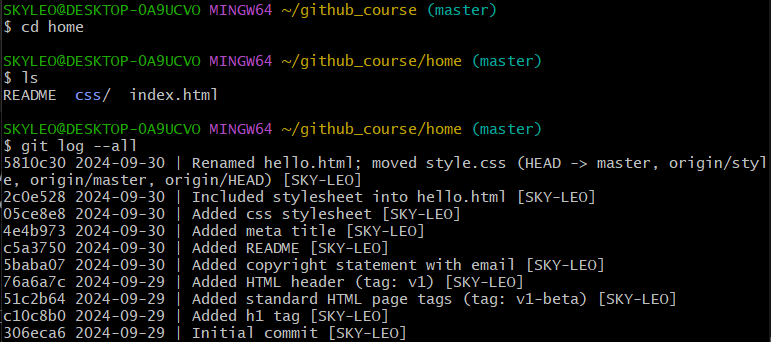


02 Создайте клон репозитория work



**31. Просмотр клонированного репозитория**

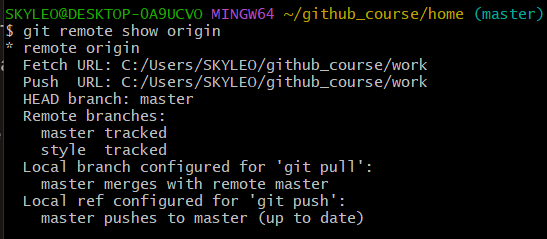
01 Посмотрите на клонированный репозиторий, просмотрите историю репозитория



**32. 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:



**33. Удаленные ветки**

Давайте посмотрим на ветки, доступные в нашем клонированном репозитории.



Как мы видим, в списке только ветка main. Где ветка style? Команда git branch выводит только список локальных веток по умолчанию.

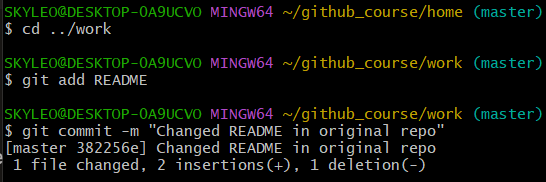
01 Список удаленных веток



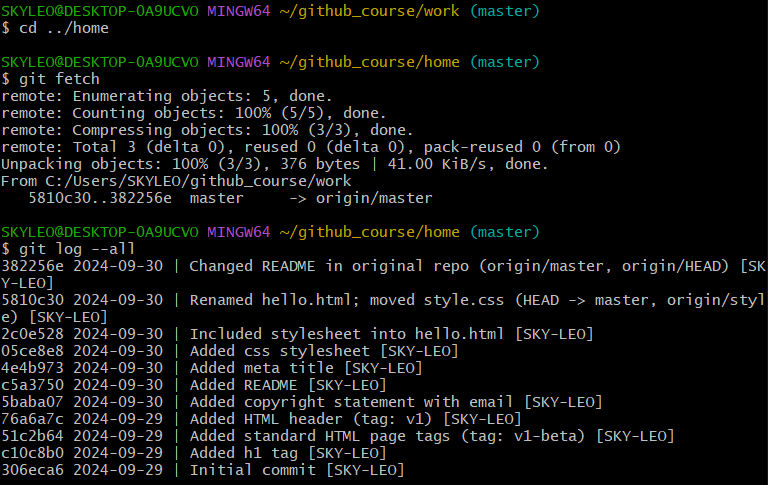
Git выводит все коммиты в оригинальный репозиторий, но ветки в удаленном репозитории не рассматриваются как локальные.

**34. Изменение оригинального репозитория**

01 Внесите изменения в оригинальный репозиторий work



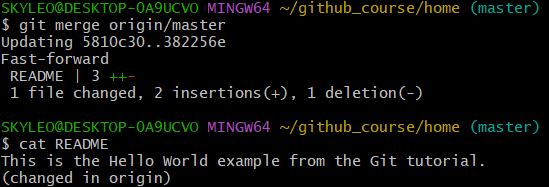
**35. Подтягивание изменений**



Команда git fetch будет подтягивать новые коммиты из удаленного репозитория, но не будет сливать их с вашими наработками в локальных ветках.

**36. Слияние подтянутых изменений**

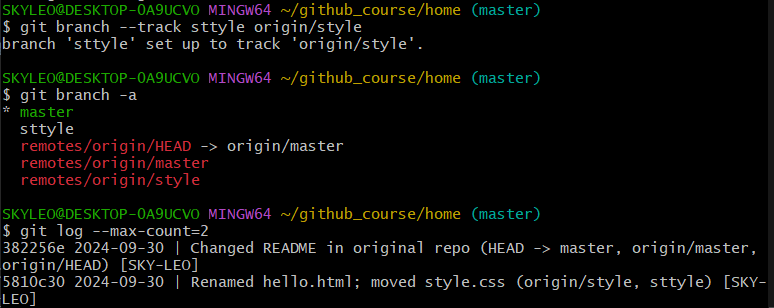
01 Слейте подтянутые изменения в локальную ветку main



Команда fetch позволяет контролировать то, что именно подтягивается и сливается в ваши локальные ветки, но для удобства существует также команда pull, которая подтягивает и сливает изменения из удаленной ветки в текущую одним вызовом.

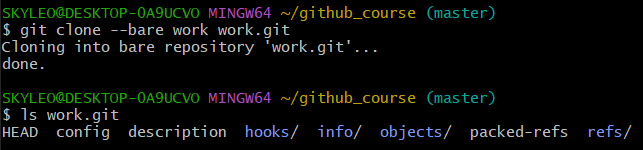
**37. Добавление ветки наблюдения**

01 Добавьте локальную ветку, которая отслеживает удаленную ветку



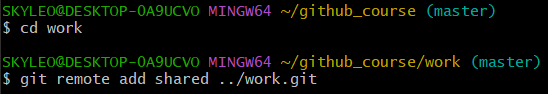
**38. Чистые репозитории**

01 Создайте чистый репозиторий



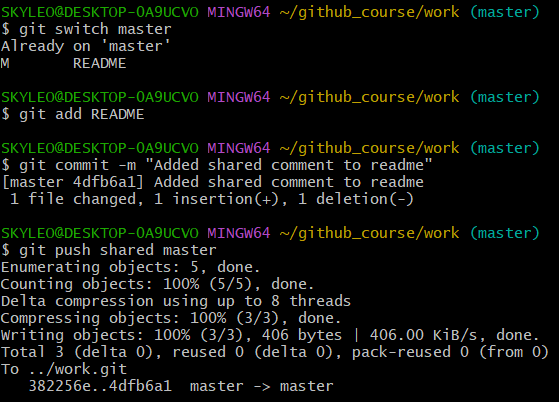
**39. Добавление удаленного репозитория**

Давайте добавим репозиторий work.git к нашему оригинальному репозиторию.



**40. Отправка изменений**

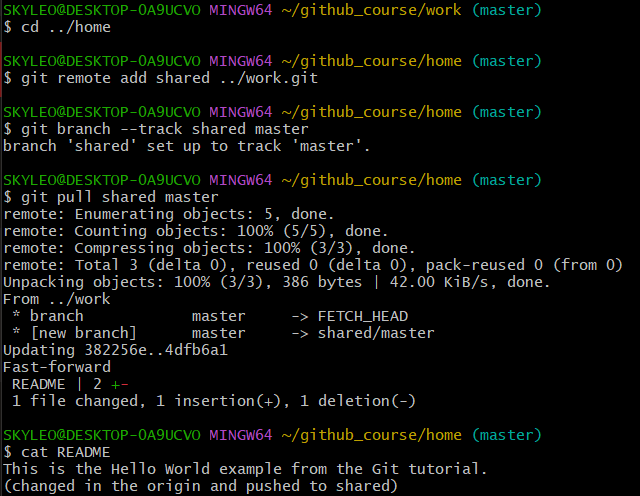
Поскольку чистые репозитории обычно располагаются на каком-либо удаленном сервере, вы не сможете туда просто зайти, дабы подтянуть изменения. Поэтому нам необходимо как-нибудь передать наши изменения в репозиторий.



Общим называется репозиторий, получающий отправленные нами изменения.

**41. Подтягивание общих изменений**

Быстро переключитесь в репозиторий home и подтяните изменения, только что отправленные в общий репозиторий.



Вывод: в ходе выполнения практического задания получены знания и базовые навыки работы с системой контроля версий Git.