

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«МИРЭА – Российский технологический университет»**

**РТУ МИРЭА**

Институт информационных технологий (ИТ)

Кафедра инструментального и прикладного программного обеспечения (ИиППО)

**ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 1 «Системы контроля версий»**

**по дисциплине «Технология разработки программных приложений»**

|  |  |
| --- | --- |
| Выполнил студент группы ИКБО-20-19 | Анваржонов Ж Т. |
|  |  |
| Принял | Иванов М.Е. |

Практические работы выполнены «09» марта 2021 г. (подпись студента)

«Зачтено» «\_\_\_»\_\_\_\_\_\_\_2021 г. (подпись руководителя)

Москва 2020

Оглавление

[Часть 1. Основные команды Git 3](#_Toc66578609)

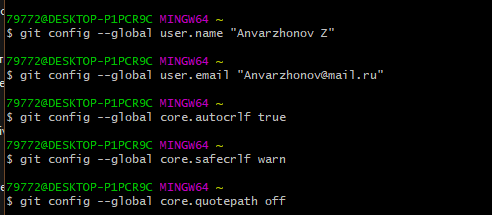
[Часть 2. Ветвление и разрешение конфликтов 14](#_Toc66578610)

[Часть 3. Системы управления репозиториями 27](#_Toc66578611)

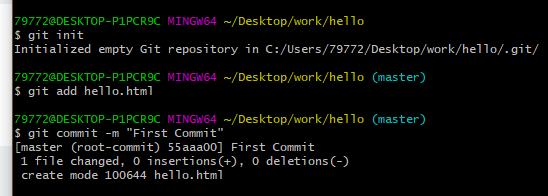
[Общее задание 31](#_Toc66578612)

[Контрольные вопросы 37](#_Toc66578613)

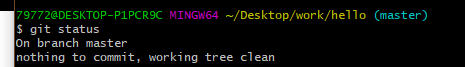
## Часть 1. Основные команды Git



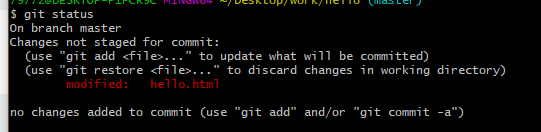
Шаг 1. Установили имя и эл почту, также параметры установки окончания строк



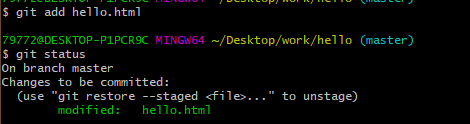
Шаг 3. Чтобы создать репозиторий из каталога, выполнили команду git init. Командой git add … добавили страницу в репозиторий и сделали коммит



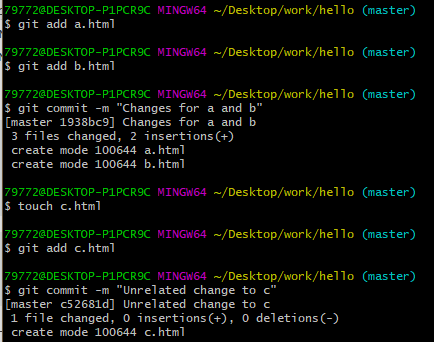
Шаг 4. Проверили текущее состояние репозитория



Шаг 5. Изменили файл hello.html, но ещё не проиндексировали

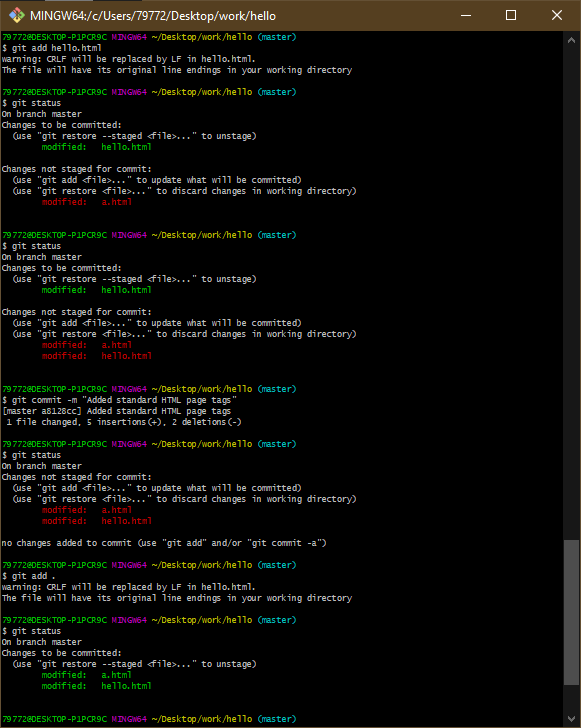


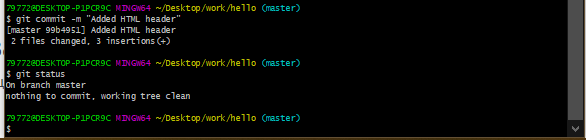
Шаг 6. Проиндексировали изменения командой git add и узнали текущее состояние



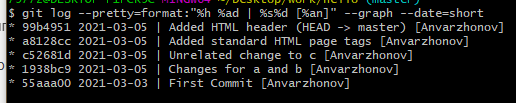
Шаг 7. Проиндексировали изменения в a.html и b.html командой git add …, при этом изменения в двух файлах сделали одним коммитом, в то как изменения в c.html идут отдельным коммитом. То есть мы разделили индексацию и коммит, чтобы настроить что идет в какой коммит

Шаг 8 пропущен

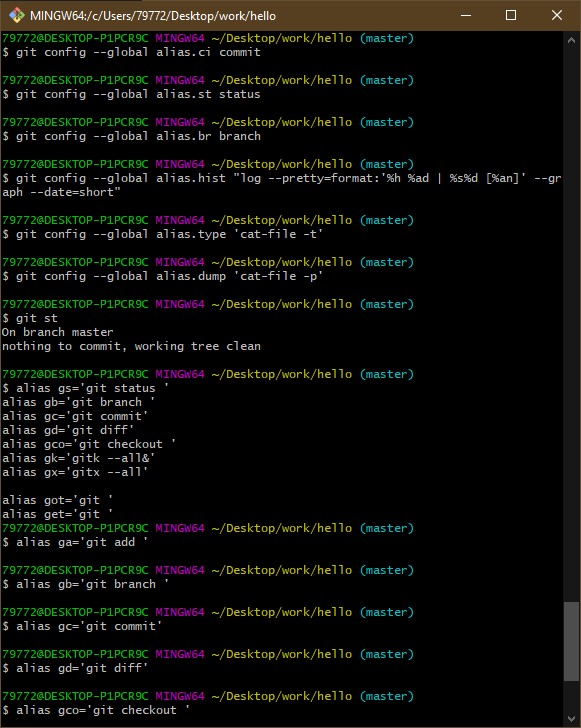




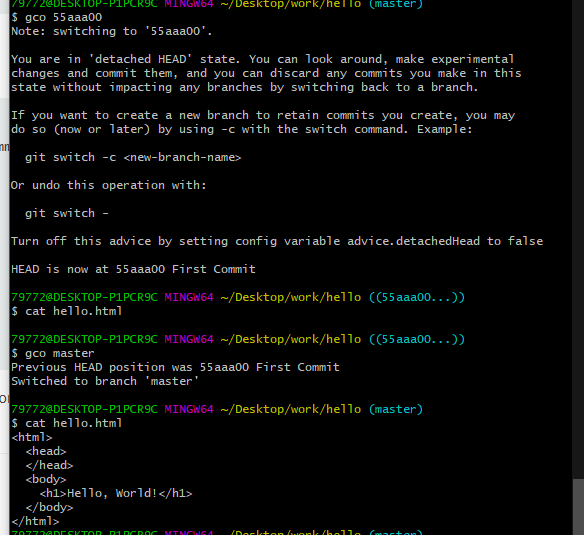
Шаг 9. Проделав шаги, мы поняли, что git работает с изменениями, а не файлами. Когда мы осуществляем команду git add file, подразумевается, что git надо отметить текущее состояние файла, коммит которого будет произведен позже. Мы не говорим git добавить файл в репозиторий!



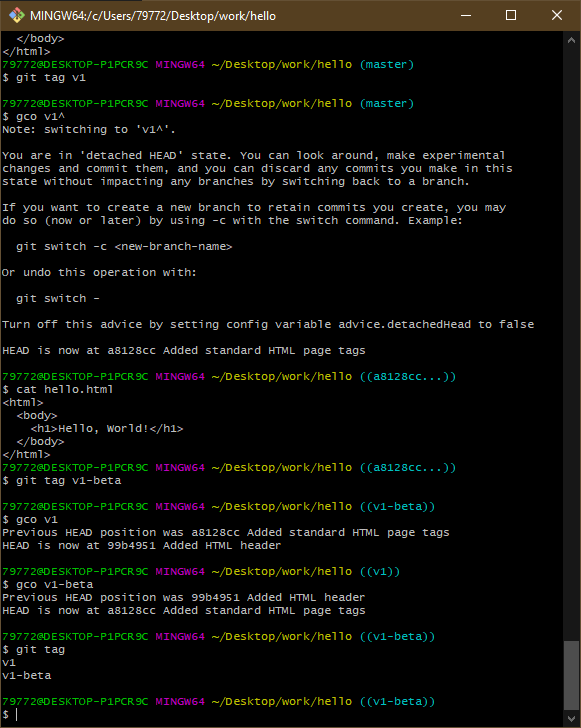
Шаг 10. Прописали формат лога, в котором легче видеть изменения и кем они были сделаны



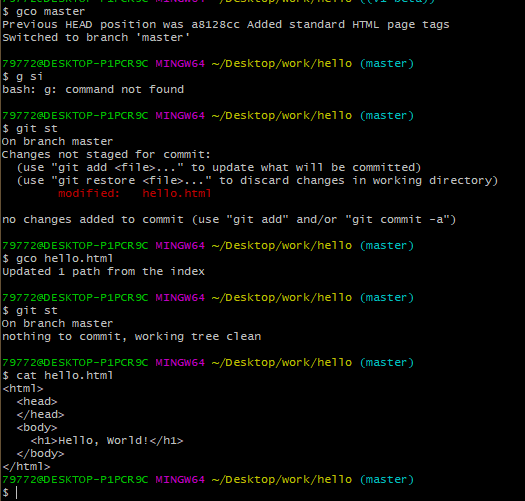
Шаг 11. Настроили алиасы и шорткаты для команды git



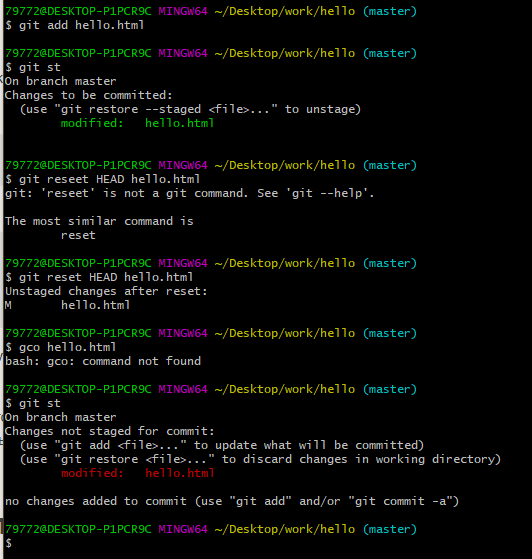
Шаг 12. Возращали рабочий каталог к любому предыдущему состоянию с помощью git checkout <hash>. «master» - имя ветки по умолчанию. Переключая имена веток, попадаешь на последнюю версию выбранной ветки



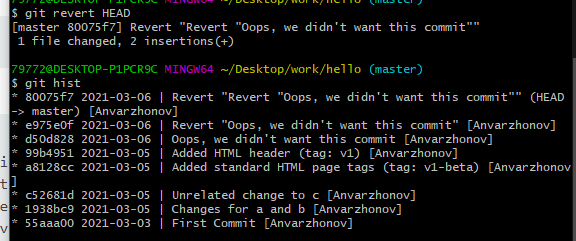
Шаг 13. Создавали теги для коммитов для использования в будущем с помощью команды git tag «…». Чтобы переключиться на предыдущую версию, надо использовать «^», обозначающий родитель v1. С помощью команды git tag можно увидеть все теги



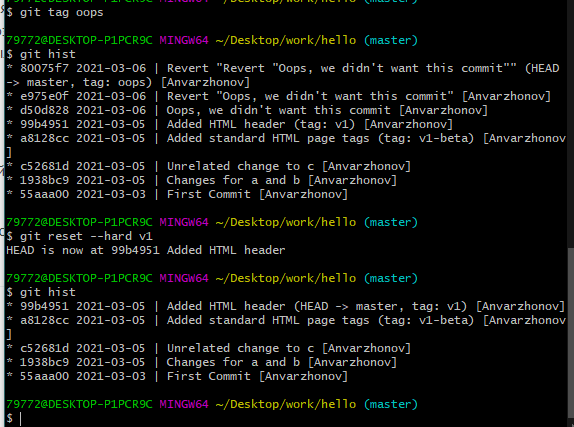
Шаг 14. Чтобы изменить последние коммиты, Когда вы изменили файл в рабочем каталоге – используем checkout



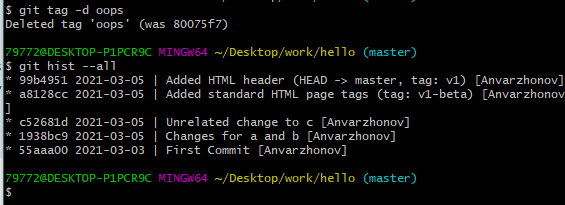
Шаг 15. Команда reset cбрасывает буферную зону к head- это очищает зону от изменений, которую мы только что проиндексировали. Команда reset не изменяет рабочий каталог. Поэтому рабочий каталог все еще содержит нежелательный комментарий. Мы можем использовать команду checkout из предыдущего урока, чтобы удалить нежелательные изменения в рабочем каталоге.



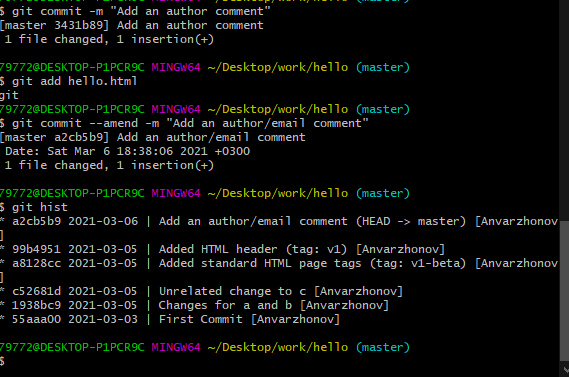
Шаг 16. Поработали с коммитом, отменяющим предыдущие. Команда Git revert «hash» позволяет отменить любые коммиты в репозиторий



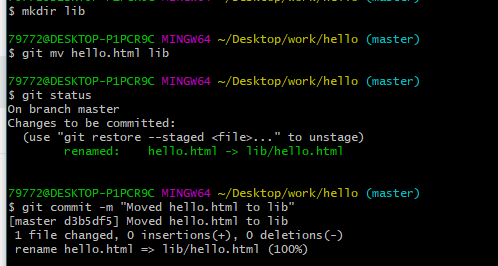
Шаг 17.Удаление коммитов из ветки. Чтобы сбросить коммиты к предшествующим, используем команду git reset –hard v1, где v1 – тег, к которому мы хотим вернуться. После этой команды ветка master будет указывать на коммит v1. –hard указывает на то, что рабочий каталог должен быть обновлен в соответствии с новым head ветки.



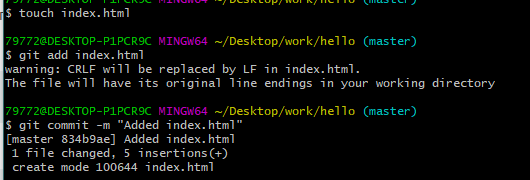
Шаг 18. При помощи git tag –d «tag» удалили тег oops. Теперь он не отображается в репозитории.



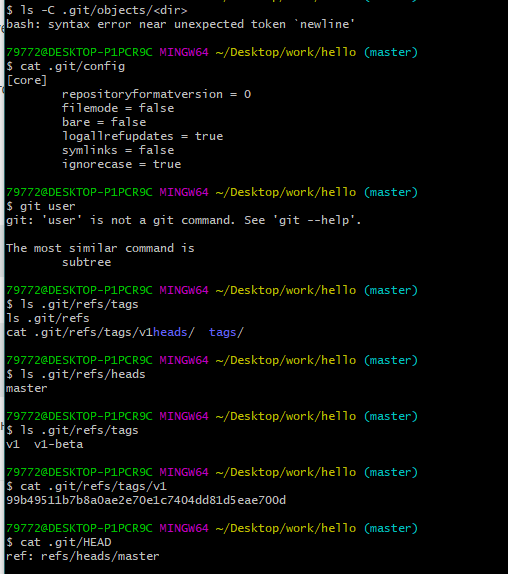
Шаг 19. Чтобы изменить сделанный коммит и не создавать новый, надо прописать команду git commit –amend –m “…”



Шаг 20. Командой mkdir lib мы создали новую папку libЧтобы переместить файл в каталог, прописали git mv hello.html lib.

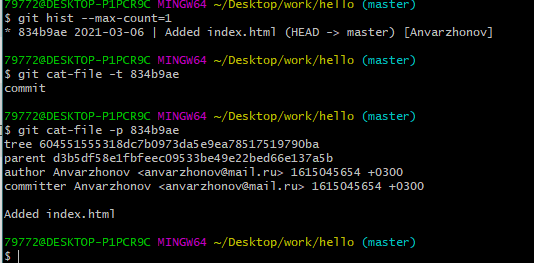


Шаг 21. Создали новый файл, добавили в репозиторий и сделали коммит.

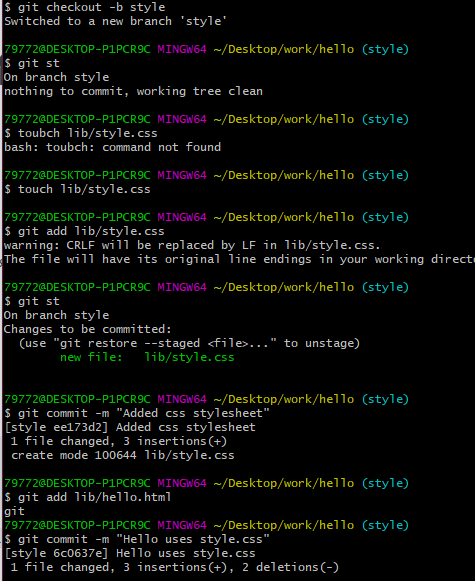


Шаг 22. Узнали о структуре каталога .git

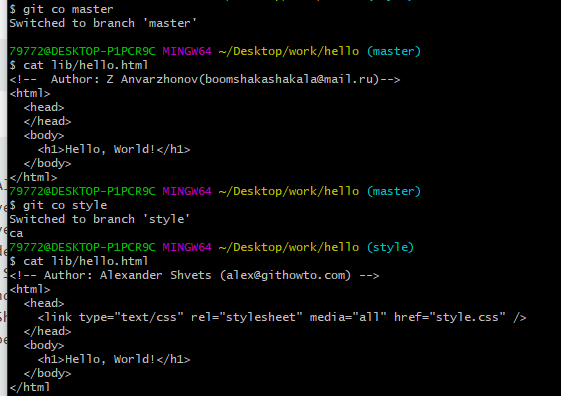
## Часть 2. Ветвление и разрешение конфликтов



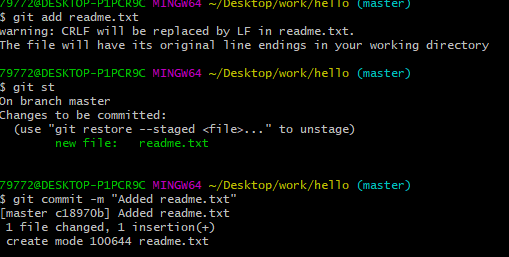
Шаг 23. С помощью команды git hist --max-count=1 показали последний коммит в репозиторий. Команды git cat-file -t <hash>, git cat-file -p <hash> можно заменить на git type и git dump



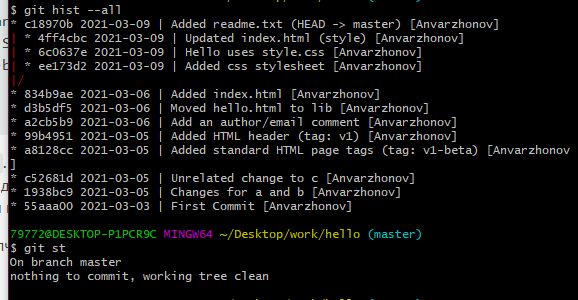
Шаг 24. Чтобы создать новую ветку, используем git checkout -b <имяветки>. Далее мы добавили три коммита в ветке style.



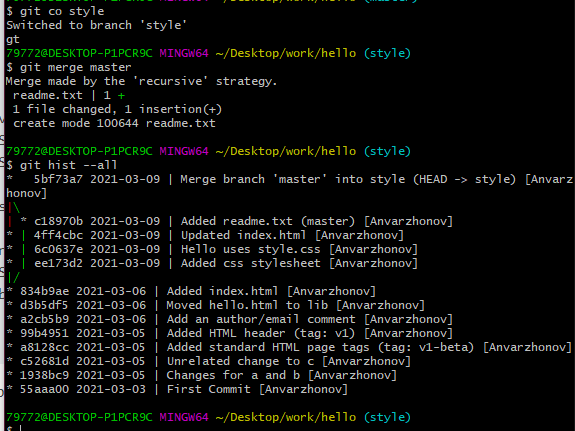
Шаг 25. Навигация по веткам. Чтобы переключиться на ветку «master», использовали команду git checkout master. Мы находимся в ветке master – это заметно, потому в файле hello.html нет стилей. Чтобы переключиться обратно на ветку style прописали снова git co style. Теперь мы в ветке style.



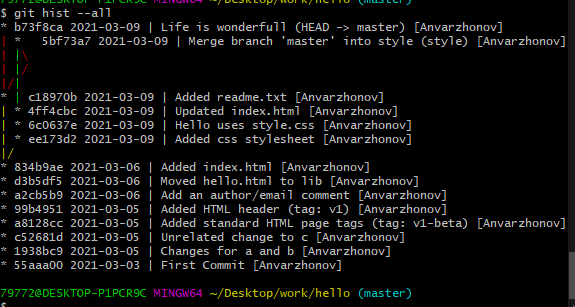
Шаг 26. Чтобы работать с несколькими ветками, надо переключаться поочередно к той ветке, которую хотим видоизменить командой git co «…»



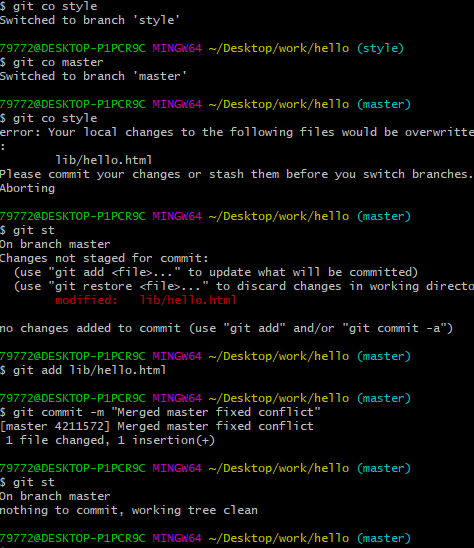
Шаг 27. Командой git hist –all, мы имеем возможность увидеть все ветки.



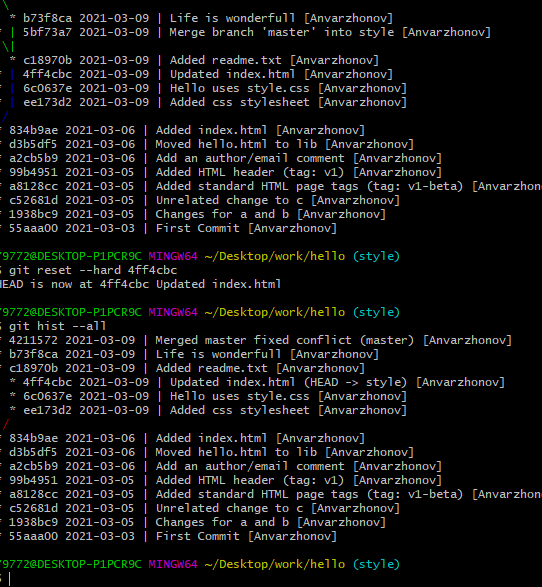
Шаг 28. Чтобы перенести изменения с двух веток в одну используем слияние веток. Для этого переключаемся командой git co style к ветке style и командой git merge master сливаем master с style.



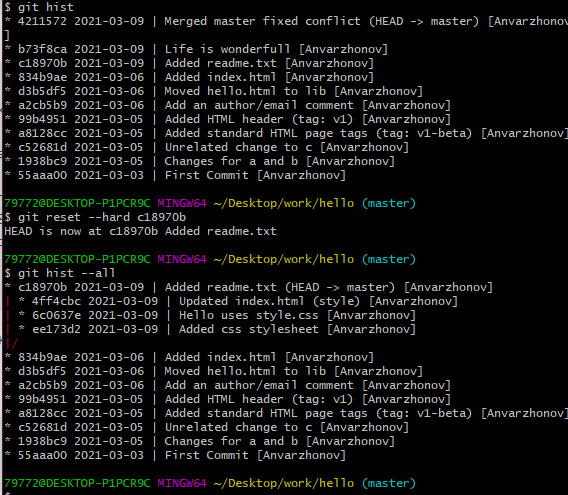
Шаг 29. (Cоздание конфликта) Создали конфликтующее изменение в ветке master. После коммита «Added README» ветка master была объединена с веткой style, но в настоящее время в master есть дополнительный коммит, который не был слит с style.



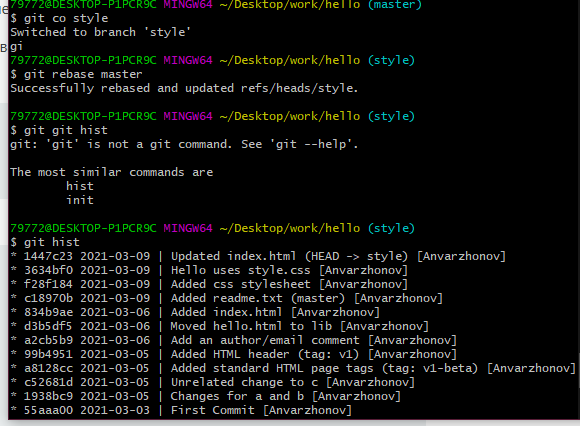
Шаг 30. (Разрешение конфликта) Чтобы разрешить конфликт, нужно вручную внести изменения



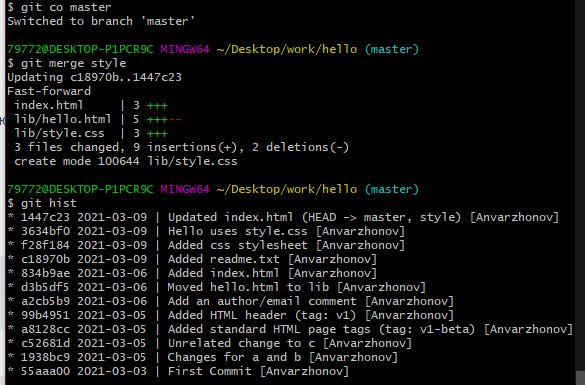
Шаг 32. Сбросили ветку style командой reset –hard “hash” до момента перед слиянием двух веток



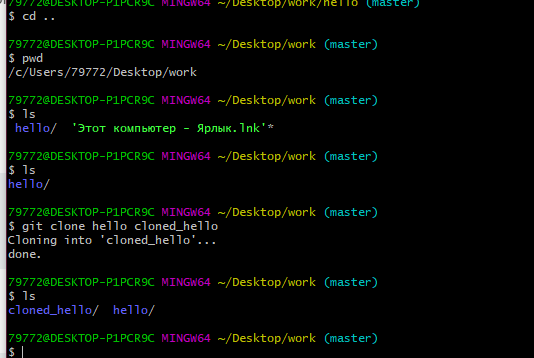
Шаг 33. Сбросили ветку master до момента перед коммитом конфликтующего интерактивного режима, то есть к коммиту Added readme.txt



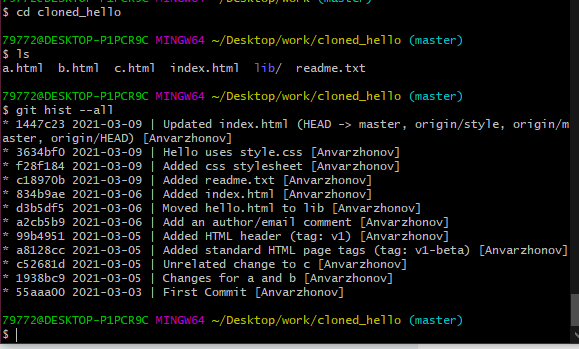
Шаг 34. (Перебазирование) Конечный результат перебазирования очень похож на результат слияния. Ветка style в настоящее время содержит все свои изменения, а также все изменения ветки master. Однако, дерево коммитов значительно отличается. Дерево коммитов ветки style было переписано таким образом, что ветка master является частью истории коммитов. Это делает цепь коммитов линейной и гораздо более читабельной. **Не следует использовать перебазирование: 1) Если ветка является публичной и расшаренной. Переписывание общих веток будет мешать работе других членов команды. 2)** **Когда важна точная история коммитов ветки (так как команда rebase переписывает историю коммитов).**

****

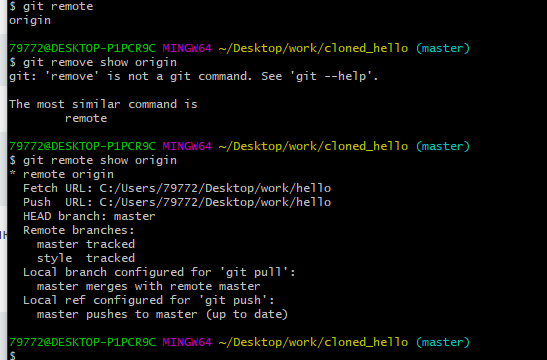
**Шаг 35. (Слияние в ветку master).** Мы поддерживали соответствие ветки style с веткой master (с помощью rebase), теперь сольем изменения style в ветку master. Теперь ветки style и master идентичны.



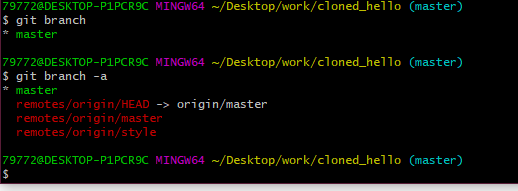
**Шаг 37. (Клонирование).** Чтобы создать клон репозитория hello прописали git clone hello cloned\_hello. Команда pwd показывает путь к каталогу. Ls – содержимое репозиториев



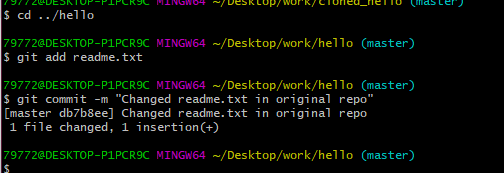
**Шаг 38. (Просмотр клонированного репозитория). Командой cs cloned\_hello перешли в папку cloned\_hello. Командой ls узнали содержимое папки.**

****

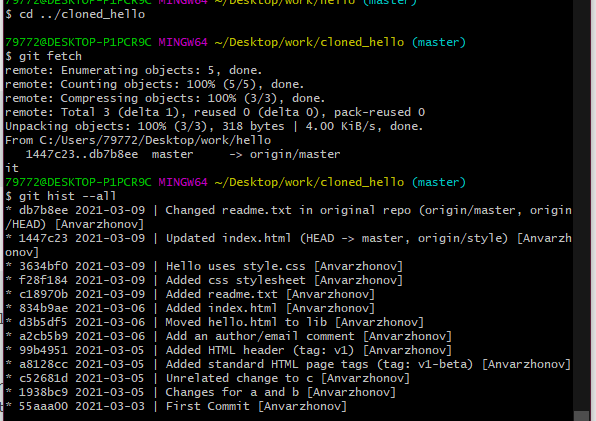
**Шаг 39.(Origin. Узнаем об именах удаленных репозиториев). Origin – «имя по умолчанию» удаленного репозитория.**

****

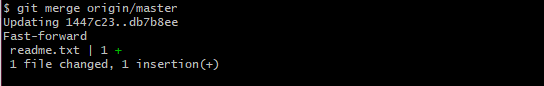
**Шаг 40(Удаленные ветки).** Команда git branch выводит только список локальных веток по умолчанию. Для того чтобы узнать все ветки, использовали команду git branch –a. Git выводит все коммиты в оригинальный репозиторий, но ветки в удаленном репозитории не рассматриваются как локальные. Если мы хотим иметь собственную ветку style, мы должны сами ее создать

****

**Шаг 41(Изменение оригинального репозитория)** Изменили оригинальный репозиторий hello. Теперь в оригинальном репозитории есть более поздние изменения, которых нет в клонированной версии.

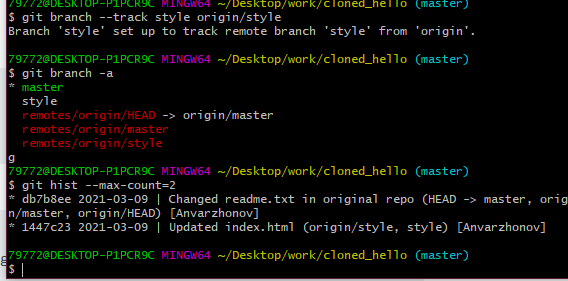
****

**Шаг 42(Извлечение изменений).** коммит «Changed README in original repo» включает в себя коммиты «origin/master» и «origin/HEAD». А в коммите «Updated index.html» видем, что локальная ветка master указывает на этот коммит, а не на новый коммит, который мы только что извлекли. **Выводом является то, что команда «git fetch» будет извлекать новые коммиты из удаленного репозитория, но не будет сливать их с вашими наработками в локальных ветках.**

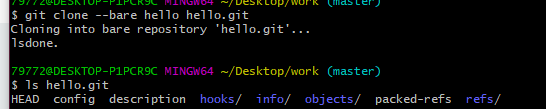
****

**Шаг 43(Слияние извлеченных изменений).** Переместили извлеченные изменения в текущую ветку и рабочий католог. Хотя команда «git fetch» не сливает изменения, мы можем вручную слить изменения из удаленного репозитория.

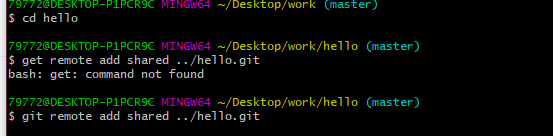
**Шаг 44(Извлечение и слияние изменений)** Команда git pull эквивалентна двум шагам: git fetch & git merge origin/master



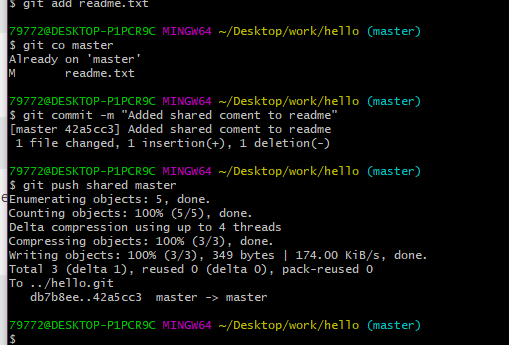
**Шаг 45(Добавление ветки наблюдения)** Добавили локальную ветку, которая отслеживает изменения удаленной ветки. Ветки, которые начинаются с remotes/origin являются ветками оригинального репозитория.



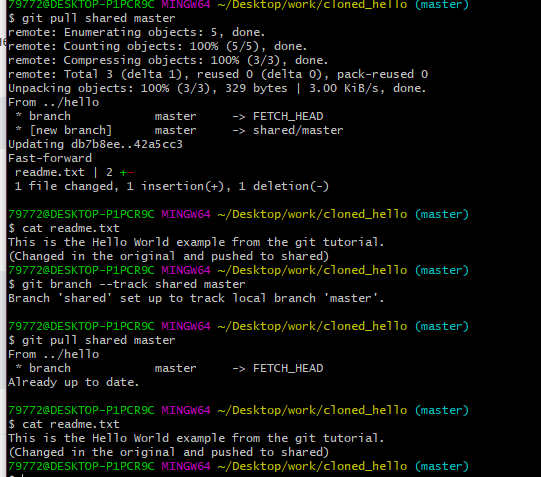
**Шаг 46(Чистые репозитории )** Чистые репозитории (без рабочих каталогов) обычно используются для расшаривания. С помощью команды git clone –bare hello hello.git создали чистый репозиторий



**Шаг 47(Добавление удаленного репозитория)** Добавили чистый репозиторий в качестве удаленного репозитория к нашему оригинальному репозиторию.

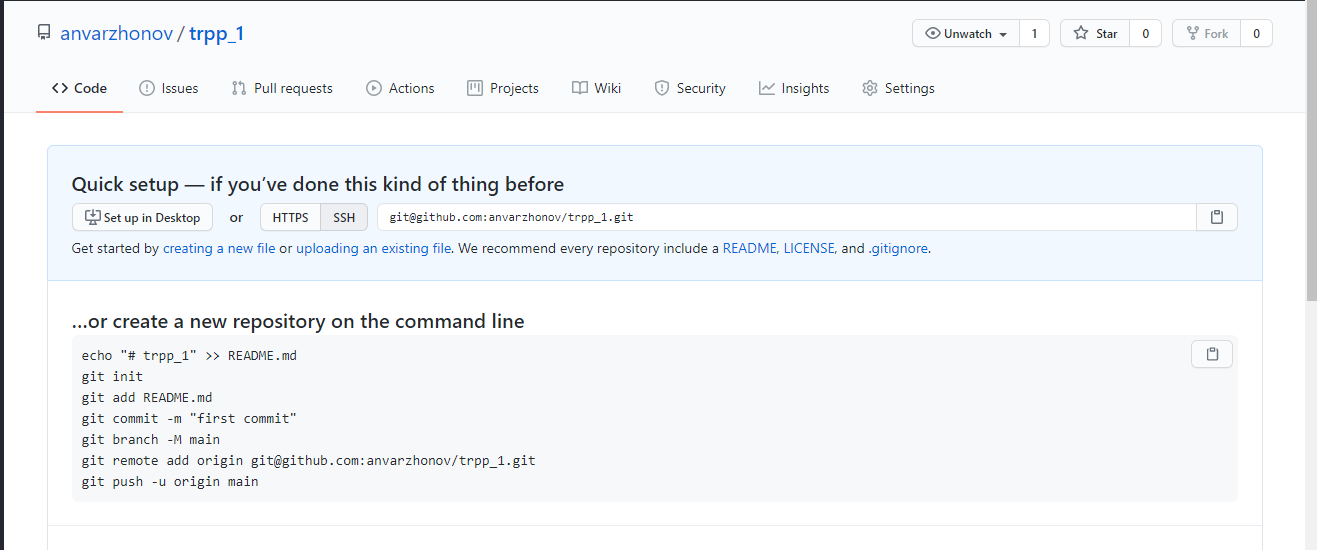
****

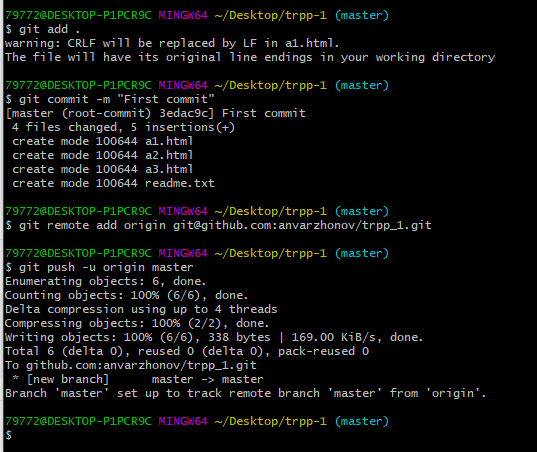
**Шаг 48(Отправка изменений)** Командой git push shared master отправили изменения в общий репозиторий. Общим называется репозиторий, получающий отправленные нами изменения.



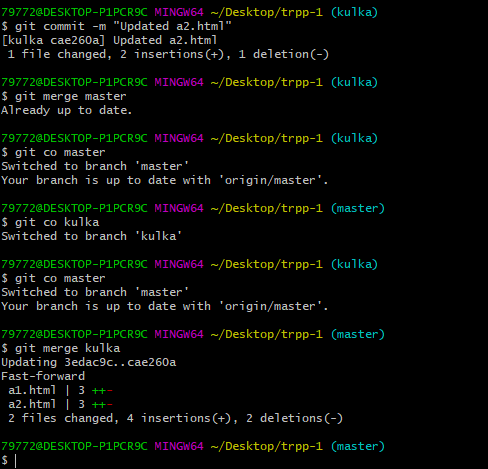
**Шаг 49(Извлечение общих изменений)** Извлекли данные, отправленные в общий репозиторий

## Часть 3. Системы управления репозиториями

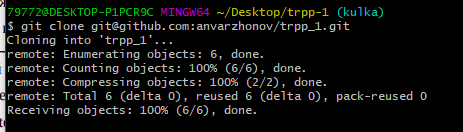
Создали репозиторий на GitHub.



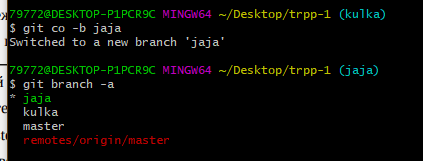
Сделали первый коммит и загрузили данные на наш удаленный репозиторий.



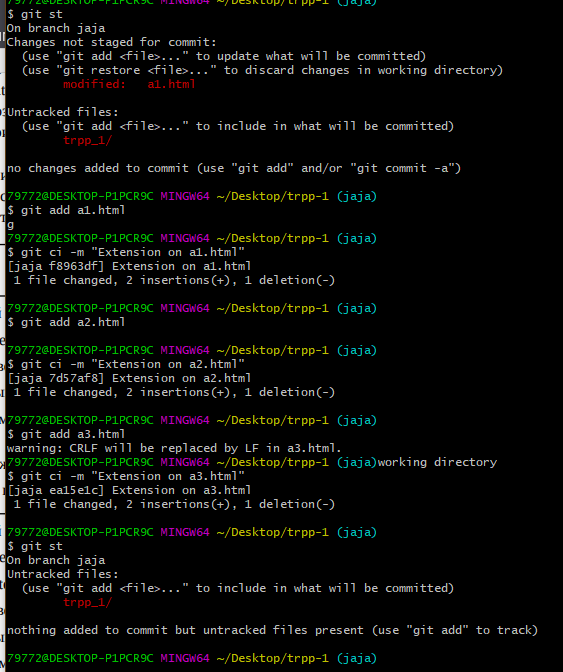
Создали новую ветку kulka, сделали несколько изменений и слили её с веткой master.



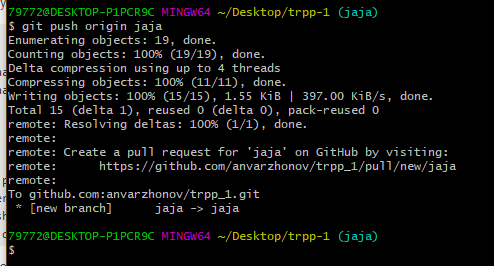
Склонировали удаленный репозиторий на локальную машину



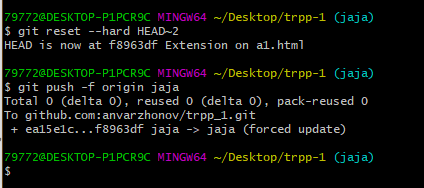
Создали новую ветку jaja и вывели все ветки командой git branch –a.



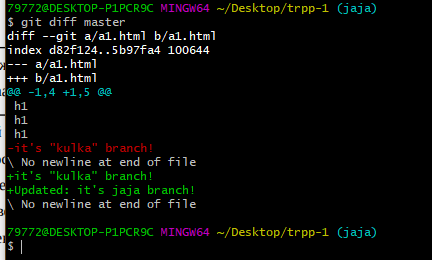
Сделали три новых коммита в ветке jaja.



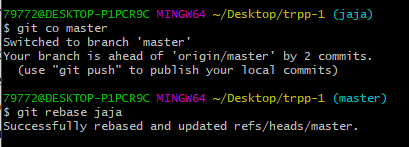
Отправили изменения в удаленный репозиторий



Откатили ветку обратно на два коммита. Добавили изменения в ветке на удаленный репозиторий

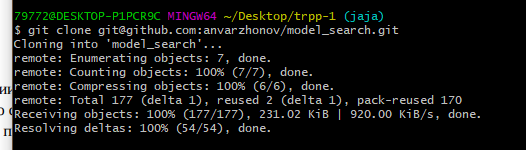


Вывели в консоле различия между ветками jaja и master.

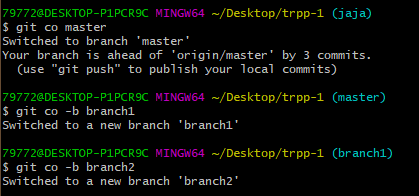


Перебазировали новую ветку на master

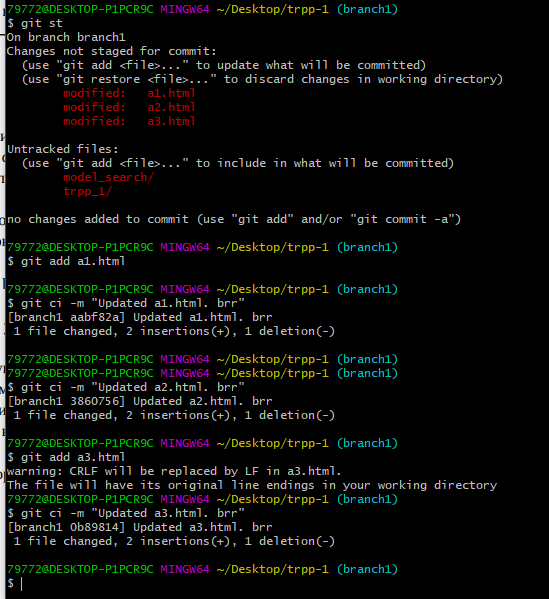
### Общее задание

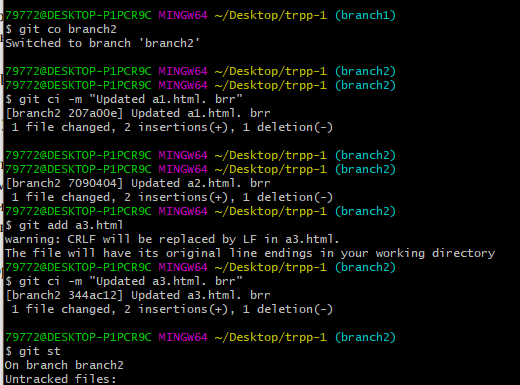


Склонировали новый реп. На локальную машину.

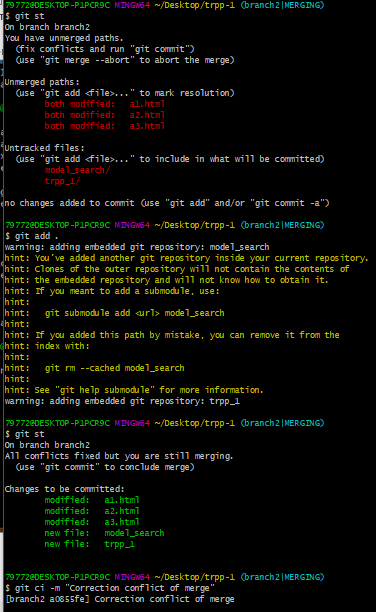


Создали на основе Master две ветки – branch1, branch2.

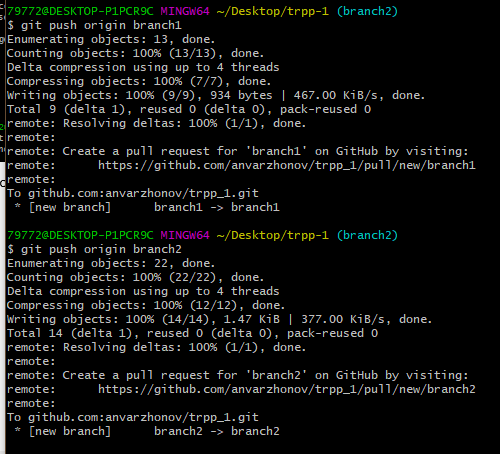




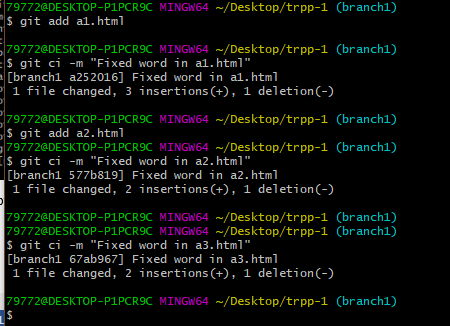
Провели 3 коммита в каждую из веток.



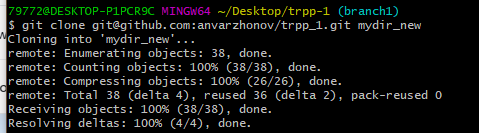
Выполнили слияние ветки branch1 в ветку branch2, разрешив конфликты



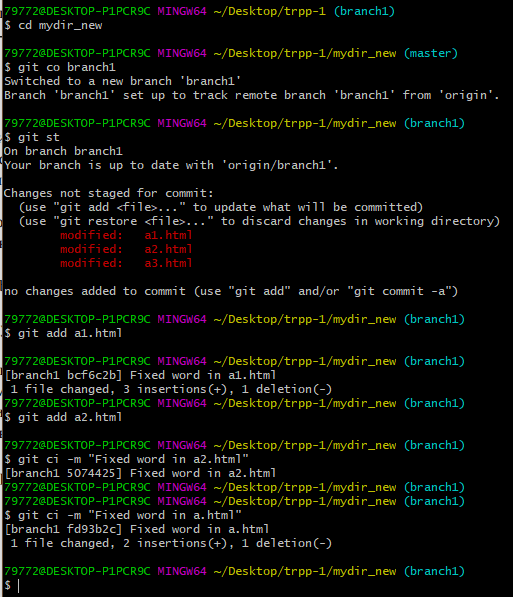
Выгрузили все изменения во всех ветках в удаленный репозиторий



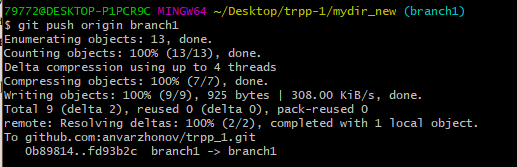
Проведили еще 3 коммита в ветку branch1



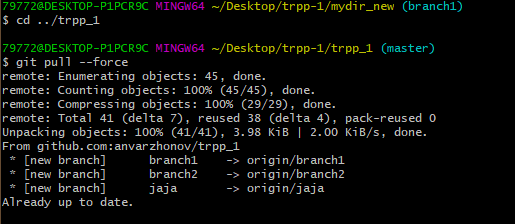
Склонировали репозиторий в новую директорию



В новом клоне репозитории сделали 3 коммита в ветку branch1



Выгрузите все изменения из нового репозитория в удаленный репозиторий



Вернулись в старый клон с репозиторием и выгрузили все изменения командой pull с опцией --force

## Контрольные вопросы

1.Какие существуют типы систем контроля версий? Приведите примеры к

каждому типу.

Наиболее распространенная **система** - это Git (вероятно, потому что GitHub предоставляет Git-репозитории бесплатно в Интернете). Есть и другие **системы** управления **версиями**, например: Mercurial, Subversion (SVN) и Perforce.

2. К какому типу систем контроля версий относится Git?

К системе управления **версиями**

3. Что такое репозиторий Git?

**Репозиторий Git** представляет собой каталог файловой системы, в котором находятся файлы конфигурации **репозитория**, файлы журналов, хранящие операции, выполняемые над **репозиторием**, индекс, описывающий расположение файлов, и хранилище, содержащее собственно файлы.

4. Что такое GitHub?

[**GitHub**](https://github.com/)**— сервис онлайн-хостинга репозиториев, обладающий всеми функциями распределённого контроля версий и функциональностью управления исходным кодом — всё, что поддерживает Git и даже больше**

5. Что такое коммит?

В системах управления версиями фиксация - это операция, которая отправляет последние изменения исходного кода в репозиторий, делая эти изменения частью основной ревизии репозитория.

6. Что такое ветка в репозитории Git?

**Ветка** в **Git** — это простой перемещаемый указатель на один из таких коммитов. По умолчанию, имя основной **ветки** в **Git** — master .

7. Что такое тег в репозитории Git?

**Теги** — это ссылки, указывающие на определенные точки в истории **Git**. Команда **git** tag обычно используется для захвата некой точки в истории, которая используется для релиза нумерованной версии (например, v1. 0.1).

8. Что такое слияние двух веток?

Это перенос кода из одной **ветки** в другую

9. Что такое конфликт в Git? Как его решить и почему они бывают?

Конфликты возникают, когда два человека изменяют одни и те же строки в файле или один разработчик удаляет файл, который в это время изменяет другой разработчик. В таких случаях Git не может автоматически определить, какое изменение является правильным. Конфликты затрагивают только того разработчика, который выполняет слияние, остальная часть команды о конфликте не знает. Git помечает файл как конфликтующий и останавливает процесс слияния. В этом случае ответственность за разрешение конфликта несут разработчики. Конфликты необходимо разрешать вручную.

10. Как отменить слияние веток, если произошел конфликт?

Для отмены слияния веток в git (git merge) есть два решения:

- удаление коммита слияния (unmerge, merge undo, merge delete)

- отмена изменений слияния (revert)

11. Для чего нужен .gitignore?

**gitignore нужен** для скрытия файлов и папок от системы контроля версий Git. Обычно скрывают конфигурационные файлы (особенно с паролями), временные файли и папки.

12. Что делает команда git status?

**Команда git status** показывает состояния файлов в рабочем каталоге и индексе: какие файлы изменены, но не добавлены в индекс; какие ожидают коммита в индексе. Вдобавок к этому выводятся подсказки о том, как изменить состояние файлов.

13. Что делает команда git add?

**Команда git add** добавляет содержимое рабочего каталога в индекс (staging area) для последующего коммита

14. Что делает команда git log?

**git log** перечисляет коммиты, сделанные в репозитории в обратном к хронологическому порядке — последние коммиты находятся вверху.

15. Что делает команда git diff?

**Команда git diff** используется для вычисления разницы между любыми двумя **Git** деревьями

16. Что делает команда git show?

Команда **git show** отображает объект в простом и человекопонятном виде. Обычно она используется для просмотра информации о метке или коммите.

17. Что делает команда git stash?

**Команда git stash** предназначена для того, чтобы поместить текущие изменения, которые вы выполнили в файлах, в отдельное хранилище, и вернуть файлы к исходному состоянию. То есть **git stash** прячет изменения в файлах и сохраняет эти изменения отдельно, чтобы потом можно было их вернуть.

18. Как добавить новую директорию в Git?

1. Открыть Git Bash.
2. Перейти в каталог, в котором вы хотите создать папку.
3. Ввести следующую команду mkdir <имя папки> и нажмите клавишу enter, чтобы создать каталог.

19. Что сделает команда "git branch" без какого-либо параметра?

**Команда git branch** — **это** своего рода "менеджер веток". Она умеет перечислять ваши ветки, создавать новые, удалять и переименовывать их. При запуске без параметров, вы получите простой список имеющихся у вас веток:

20. Чем отличаются команды "git push" и "git pull"?

**Команду git push** можно рассматривать как **команду** «выгрузки», а **команды** git fetch и **git pull** — как **команды** «загрузки». После того как наборы изменений перемещены посредством «загрузки» или «выгрузки», в месте назначения можно интегрировать изменения, выполнив их слияние **командой** git merge .

21. Что означает статус файла untracked в выводе команды git status?

**Статус Untracked** означает, что **Git** видит файл, которого не было в предыдущем снимке состояния (коммите); **Git** не станет добавлять его в ваши коммиты, пока вы его явно об этом не попросите.

22. Что означает статус файла new в выводе команды git status?

Это значит что добавленный файл теперь отслеживаемый и добавлен в индекс

23. Что означает статус файла modified в выводе команды git status?

Это значит что файл был модифицирован

24. Чем отличается master и origin/master?

**master** - это ваш местный ветка. **origin master** -это ветвь **master** в удаленном репозитории с именем **origin** . **origin**/**master** -это ваша локальная копия **origin master**

25. Как узнать, кто автор строчки в файле, используя систему Git?

Через git hist

26. Как отменить действие команды "git add" на файл?

с помощью git reset <file>

27. Как привести измененный файл в начальное состояние (до изменения)?

1. Выяснить обозначение коммита через git log или подобную команду.

2. Следует дать команду:

git checkout <hash> <имя\_файла>

28. Как сделать ветку с названием my\_branch?

C помощью команды git checkout –b my\_branch.

29. Как сделать коммит для ветки my\_branch?

Переключиться на ветку my\_branch с помощью git checkout и сделать коммит в этой ветке

30. Как удалить локальную ветку my\_branch?

git branch –D my\_branch

31. Как исправить ошибку "fatal: The current branch my\_branch has no upstream

branch", возникающую при вводе git push?

32. Как удалить все untracked файлы?

[git-clean](https://git-scm.com/docs/git-clean) -n - удаление неотслеживаемых файлов из рабочего дерева

33. Что такое форк репозитория?

**Fork** - это копия **репозитория**

34. Что такое pull request?

**Pull Request** — это запрос на вливание изменений из вашей ветки в основную ветку исходного репозитория