4Министерство образования и науки Российской Федерации

Федеральное государственное образовательное учреждение высшего образования

«Алтайский государственный технический университет им. И. И. Ползунова»

Факультет информационных технологий

Кафедра прикладной математики

Отчет защищен с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

«\_\_\_\_\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г.

Отчет

по лабораторной работе № 1

*«Git + GitHub»*

по дисциплине «Программирование»

Студент группы ПИ-92:

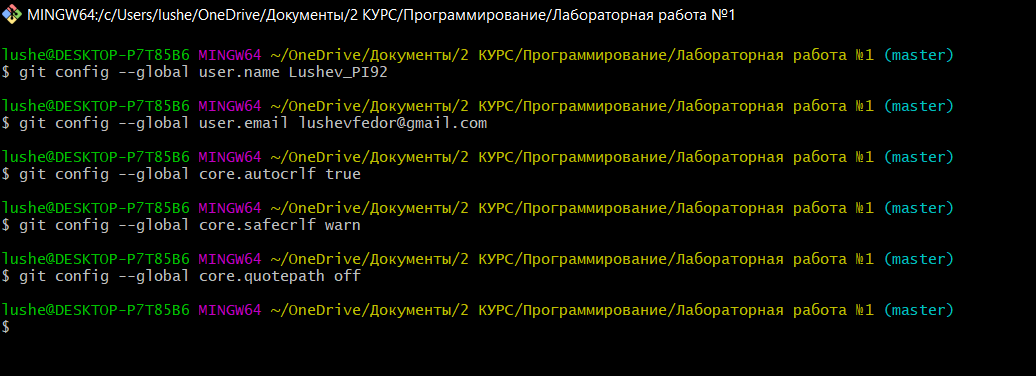
Лушев Ф.Е

Преподаватель Троицкий В.С.

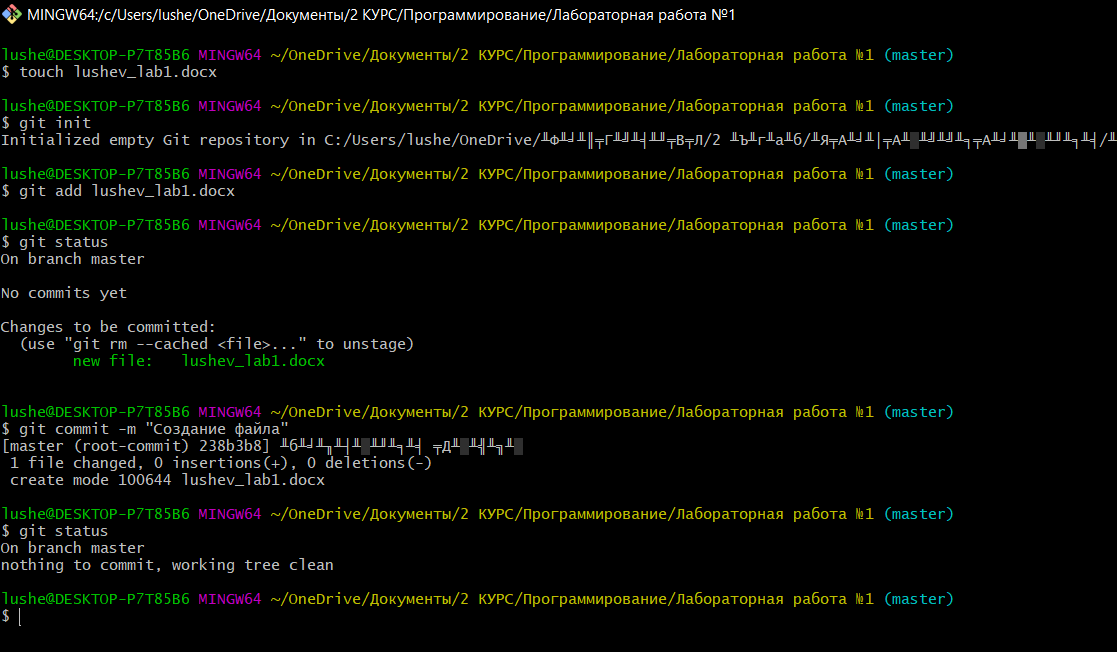
Барнаул 2020

## Настройка Git, создание репозитория и отчёта.

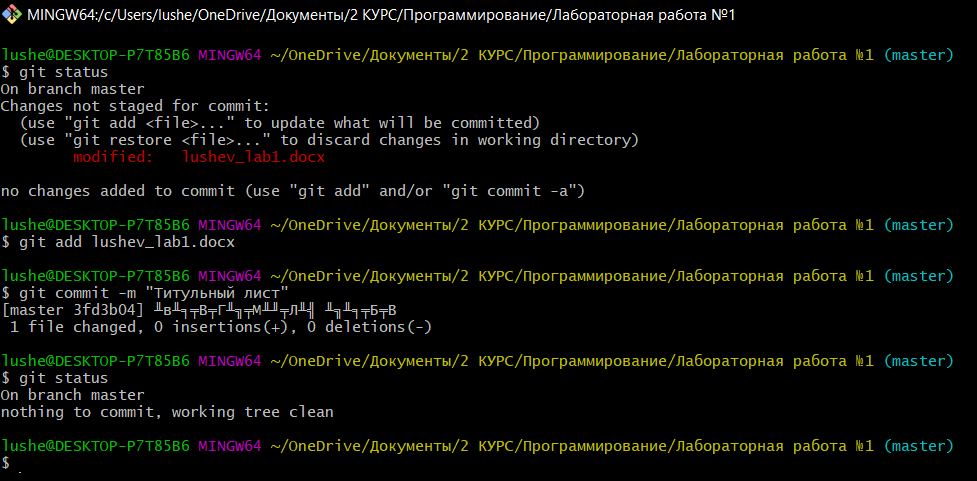
1. С помощью команды “cd” ставим Git в нужную нам директорию, затем производим первоначальную настройку:



1. Создаем отчёт с помощью команды “touch”, затем создаем репозиторий с помощью команды “git init” и добавляем в него наш отчёт с помощью команды “add”. Пишем первый коммит «Создание файла»:



1. Формируем титульный лист и создаем соответствующий коммит:

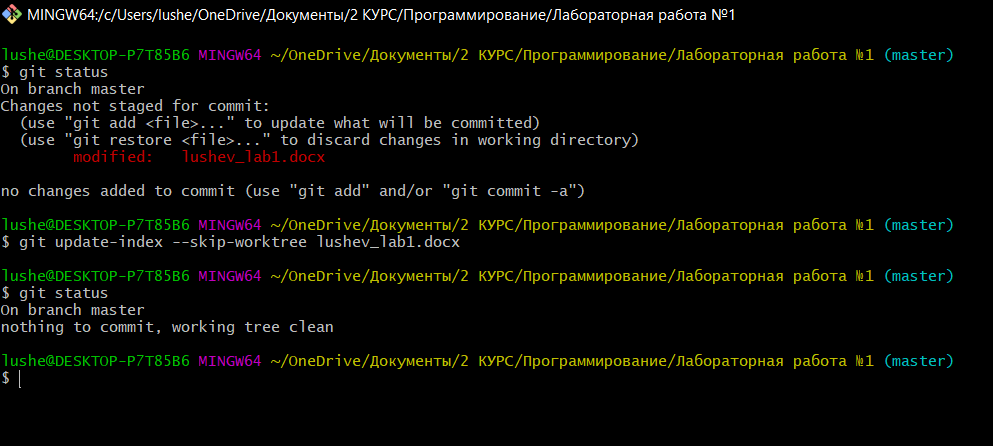


## Игнорирование, удаление, сравнение и перемещение файлов.

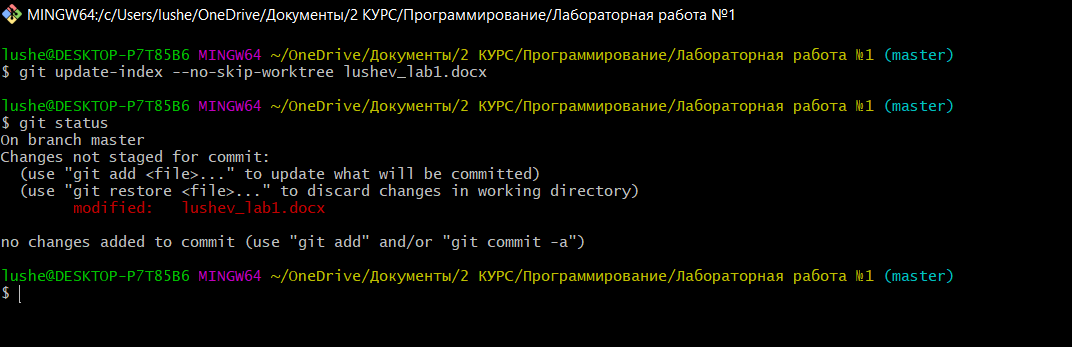
Игнорирование:

1. В отчёт внесено неиндексированное в Git изменение, что показывает команда “git status”.

С помощью команды “git update-index –skip-worktree” мы игнорируем изменения, вносимые в файл и они больше не отображаются в “git status”:



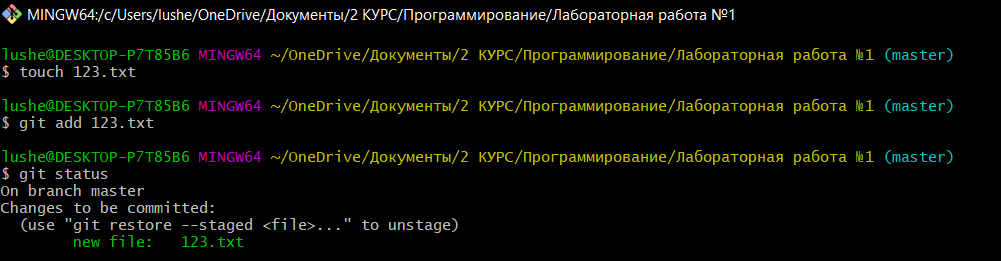
1. Затем с помощью команды “git update-index –no-skip-worktree” восстанавливаем учёт изменений в отчёте и они снова начинают отображаться в “git status”. После убираем изменения, используя команду “git restore”:



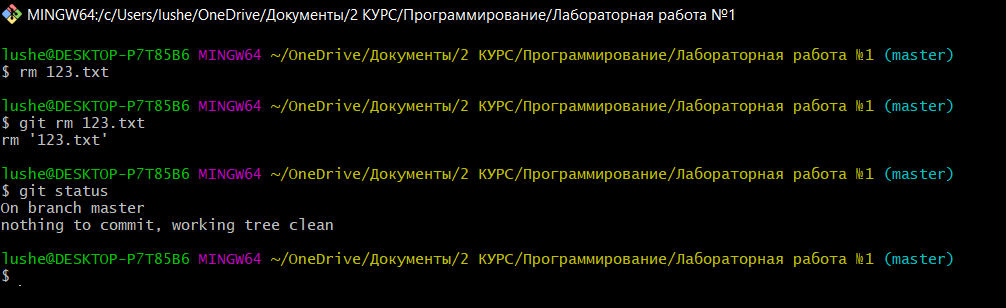
Удаление:

1. Сперва создадим в этой же директории новый файл “123.txt” и добавим его в репозиторий.

Так как мы не писали коммит, то файл отображается как добавленный в “git status”:

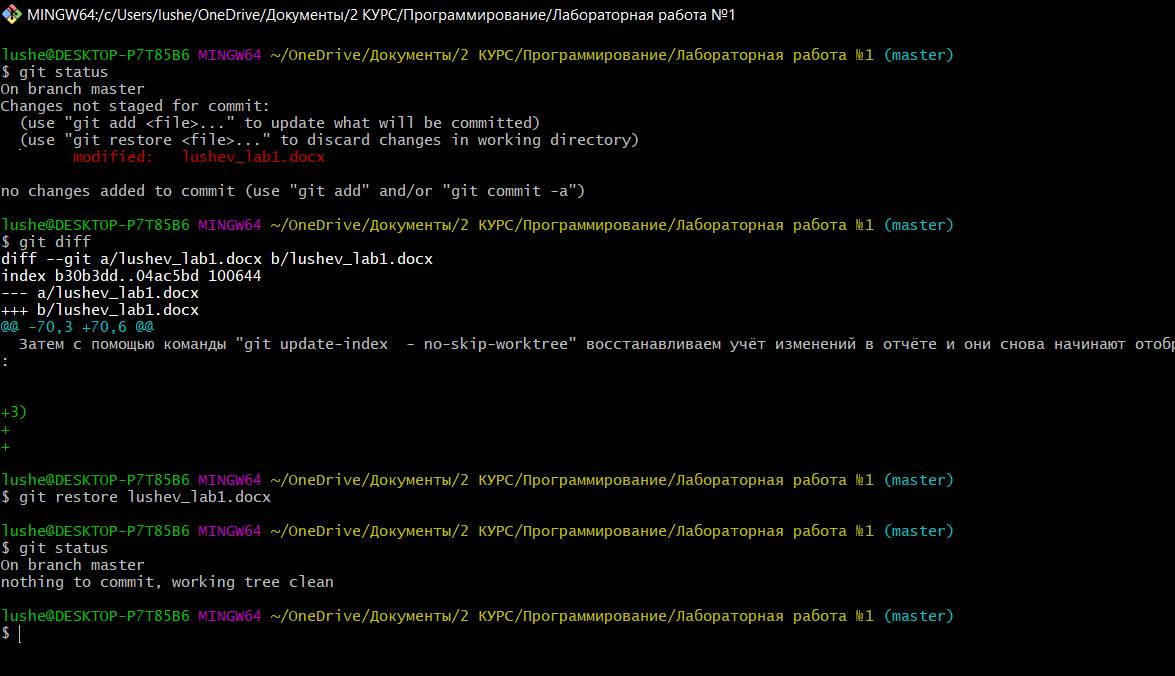


1. С помощью команды “rm” мы сперва удалим сам файл из директории, а затем с помощью команды “git rm” удалим файл из репозитория Git. Теперь “git status” снова не видит никаких изменений:



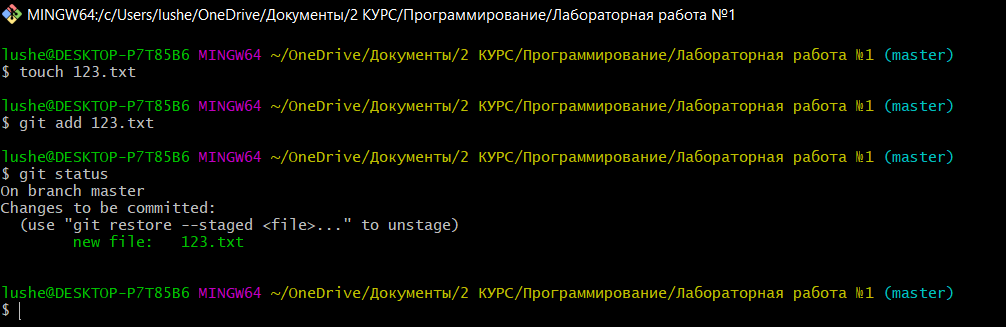
Сравнение:

1. Добавим к отчёту неиндексированное изменение, а именно “3)” в конце. Далее используя команду “git diff” мы увидим, что именно и в каком месте файла было изменено. Затем уберем изменение с помощью “restore”:

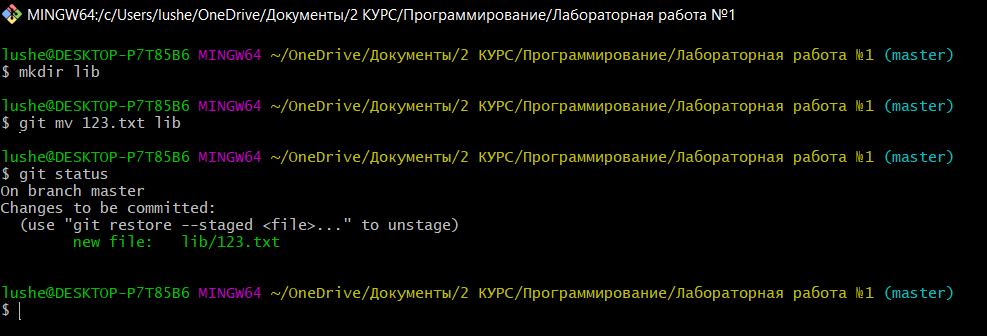


Перемещение:

1. Создадим файл 123.txt и добавим его в наш репозиторий:

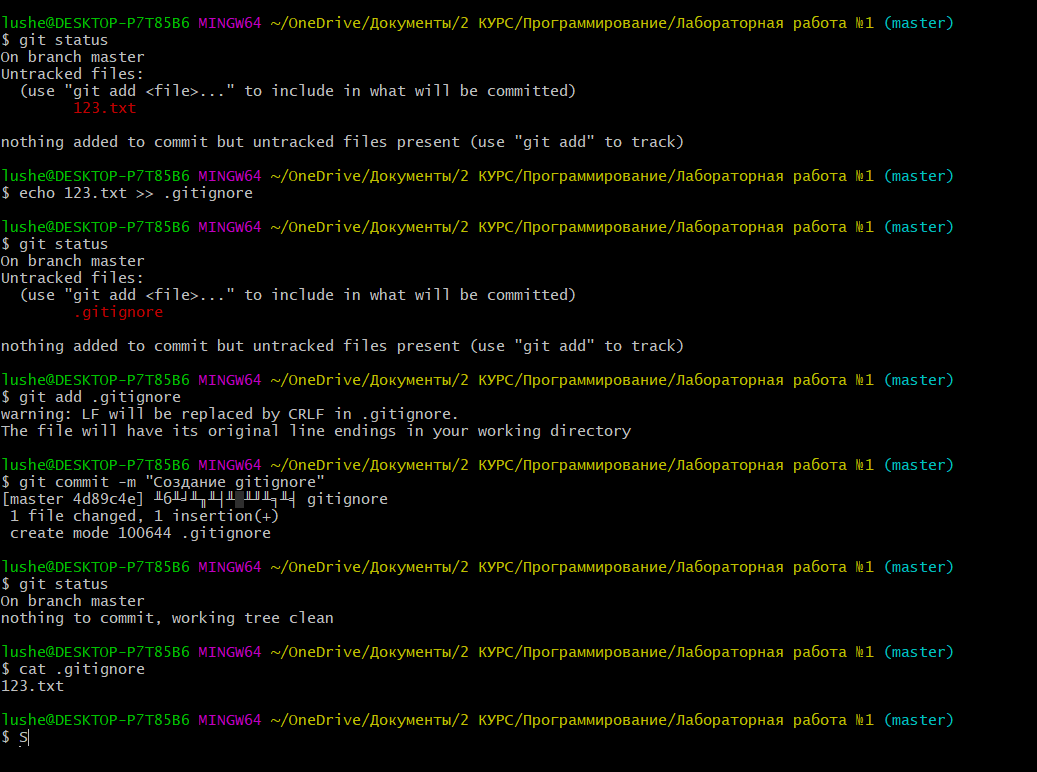


1. Теперь создадим новую директорию “lib” и переместим туда наш файл с помощью команды “git mv”. В “git status” теперь отображается, что файл находится в другой директории:

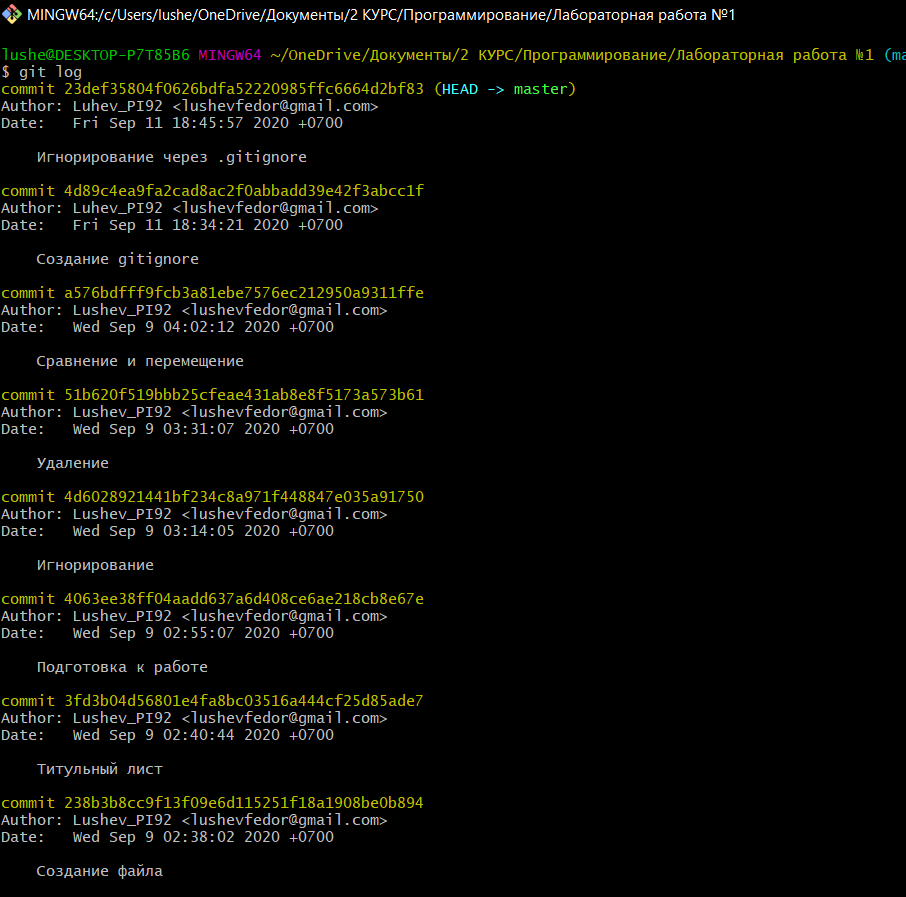


Игнорирование через .gitignore:

1. Создадим файл «123.txt», который git будет видеть как неиндексированный. Затем с помощью команды «echo» создадим файл «.gitignore» и сразу запишем туда «123.txt». Теперь добавим в репозиторий файл «.gitignore» и сделаем соответствующий коммит. Теперь Git не видит файла «123.txt», хотя он по прежнему находится в директории. Так де с помощью команды «cat» мы можем посмотреть содержимое файла «.gitignore».



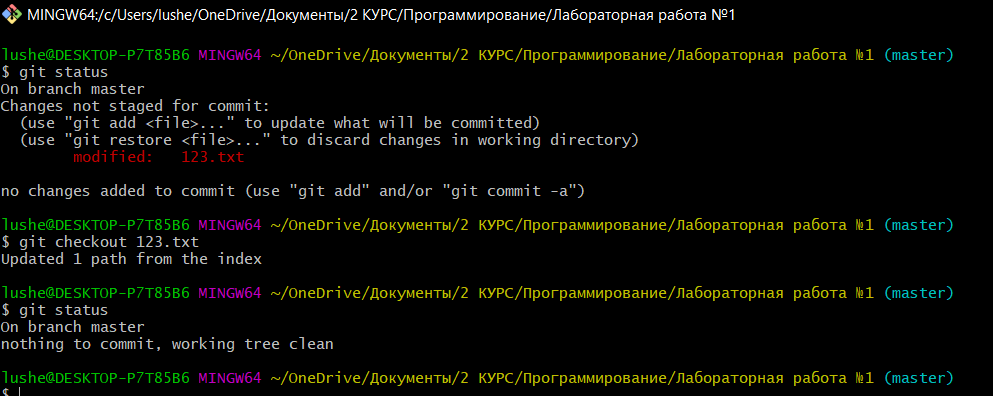
1. Просмотр коммитов:
2. Просмотреть коммиты можно с помощью команды «git log».



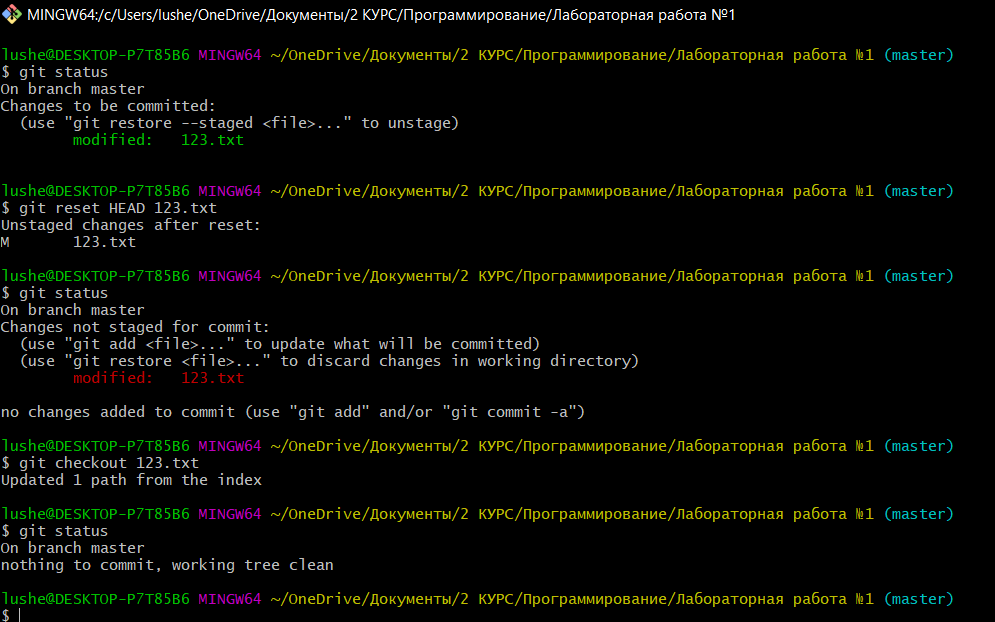
1. Отмена изменений и работа с метками.

Отмена изменений:

1. Внесем изменения в файл «123.txt». Для отмены неиндексированных изменений следует использовать команду «checkout».



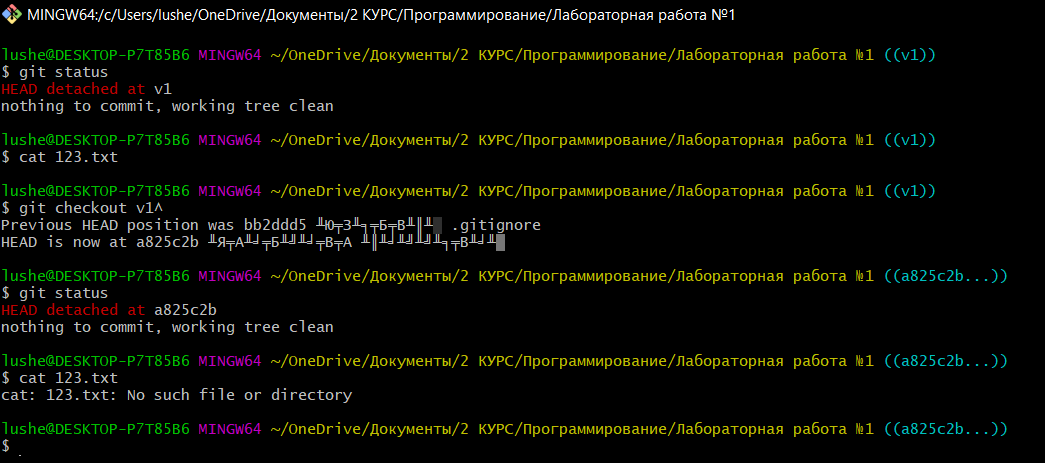
1. Для отмены проиндексированных изменений перед коммитом используется команда «reset HEAD», а затем «ckeckout».



Работа с метками:

1. Используя команду «tag» назовем текущую версию «v1». В данной версии файл «123.txt» пустой и находится в директории с проектом. Убедиться в этом мы можем с помощью команды «cat», мы получим пустою строку, так как в файле ничего нет.

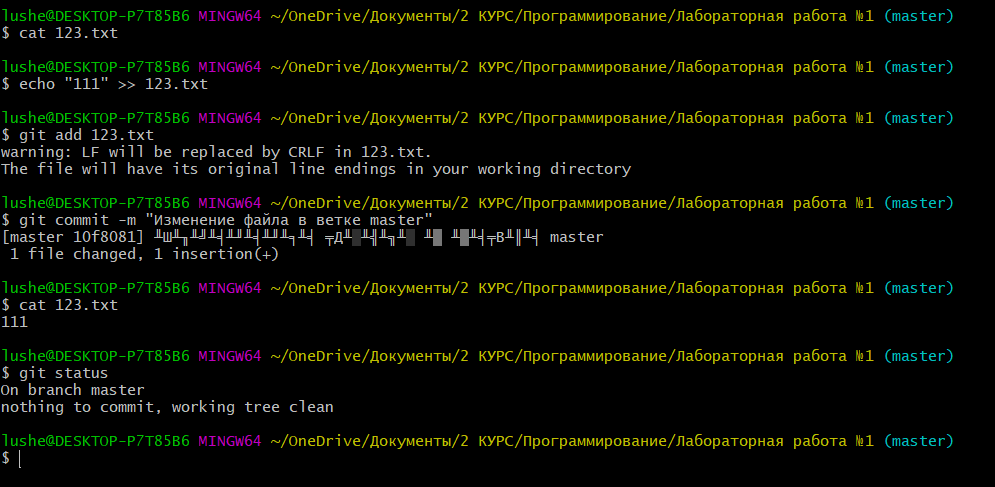
После перейдем к предыдущей версии с помощью команды «checkout v1^», в которой файл «123.txt» еще находится в .gitignore. Теперь при попытке просмотреть его содержание получим ошибку, так как Git этого файла не видит, из-за более старой версии проекта.



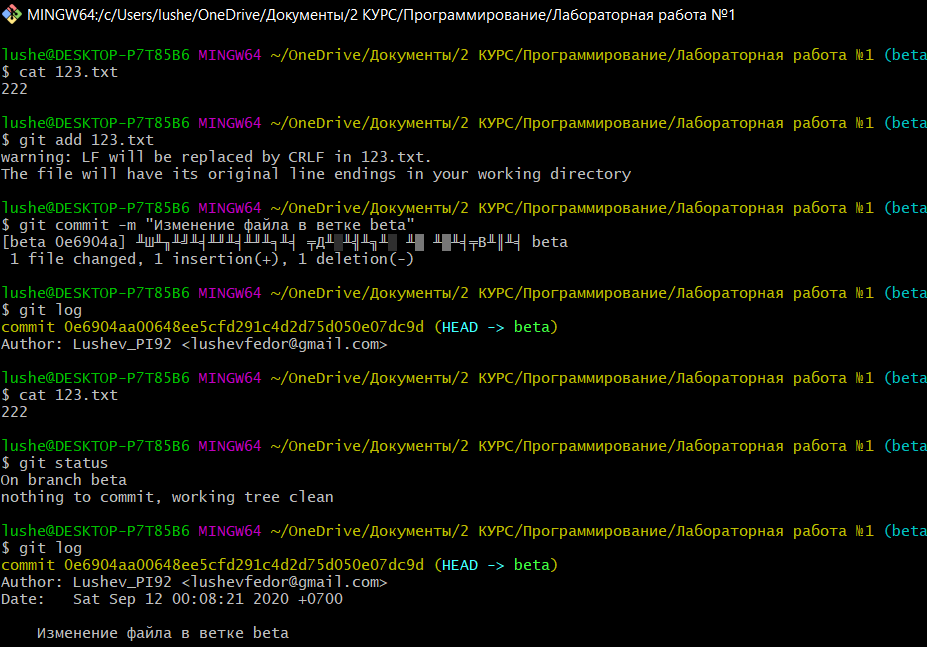
1. Работа с ветками.

Создание и перемещение:

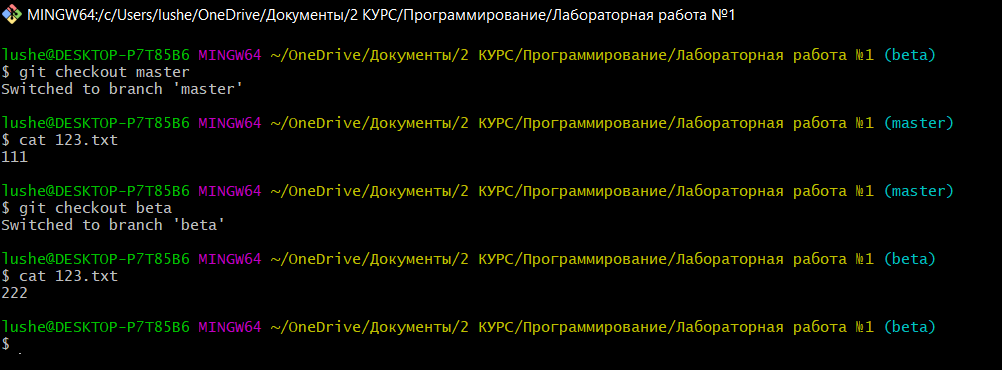
1. Сперва изменим файл «123.txt» в исходной ветке master. Создадим в этой ветке соответствующий коммит. Затем используя команду «checkout -b» создадим вторую ветку beta.



1. После с помощью команды «checkout beta» перейдем в новую ветку, где снова изменим файл «123.txt» и создадим на это коммит. С помощью команды «log» так же можно посмотреть какие коммиты и в какой ветке были сделаны.

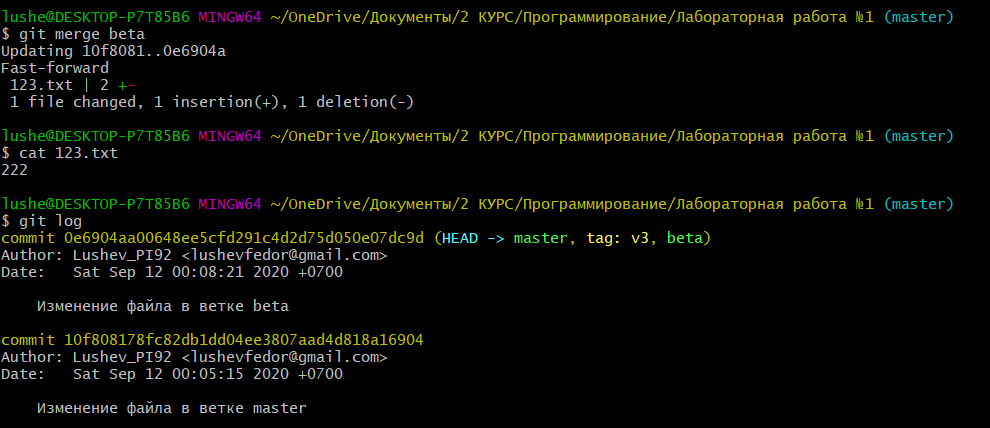


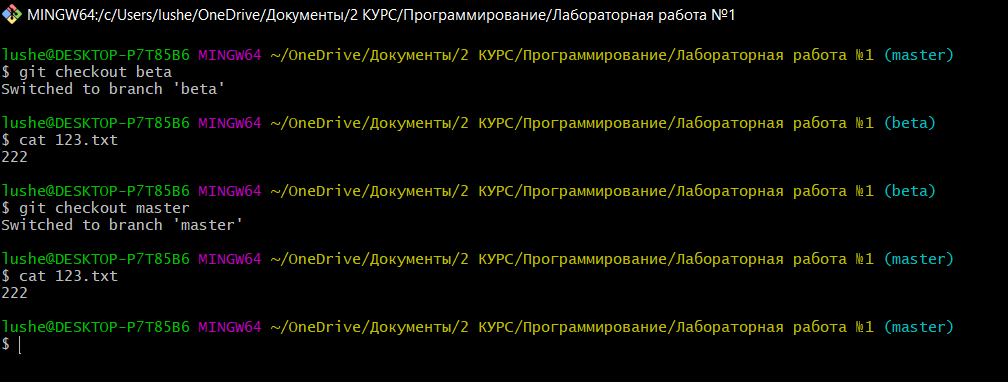
1. Теперь наглядно видно, зачем нужны разные ветви: файл «123.txt» принимает в разных ветках разные значения, хотя по факту в директории файл один с единственным значением.



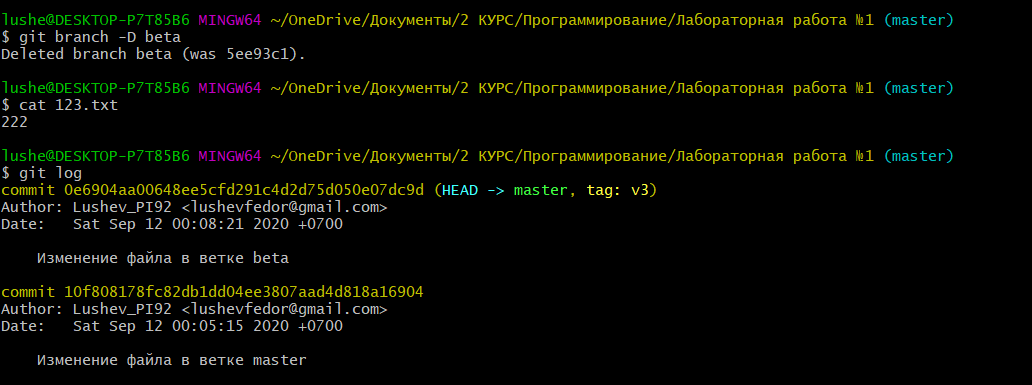
Объединение и удаление:

1. Для объединения ветвей используется команда «merge». Теперь по логу видно, что последние коммиты относятся сразу к двум веткам, однако файл «123.txt» принял одно единственное значение.



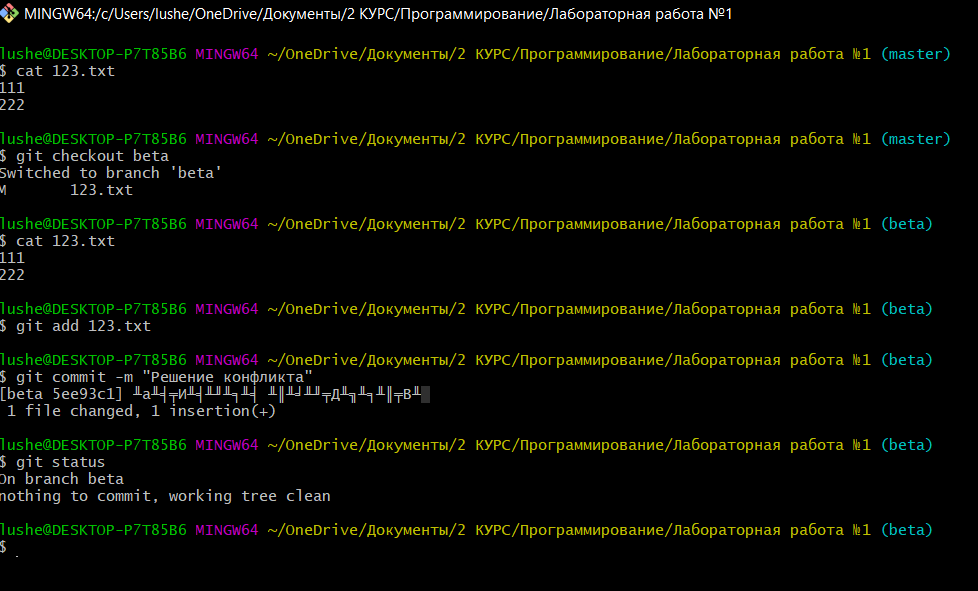


1. Для удаления ветки используется команда «branch -D». Таким образом мы можем удалить ветку beta. Теперь все коммиты относятся к ветке master.



Конфликт и разрешение:

1. В качестве примитивного конфликта можно рассмотреть ситуацию выше со слиянием веток, когда файл «123.txt», имеющий разные значения в ветках, после слияние принял только одно из этих значений. Исправить такой конфликт проще всего вручную, изменив «123.txt» и сделав коммит.



1. Git stash.
2. С помощью Git stash можно охранить неиндексированные изменения файла, чтобы потом вернуться к ним в любой момент. На примере видно, что файл «123.txt» изначально содержал «222», а после его изменили на «222333», затем с помощью команды «git stash» изменения сохранились и перестали отображаться (в том числе и в «git status»). Теперь, когда работа с файлом окончена и нужно вернуть изменения, которые мы сохранили, воспользуемся командой «git stash pop».

