Перегудова Олеся

**День 1. Изучение Git и GitHub**

**Цель:** познакомиться с работой Git и GitHub, изучить основные команды и их применение.

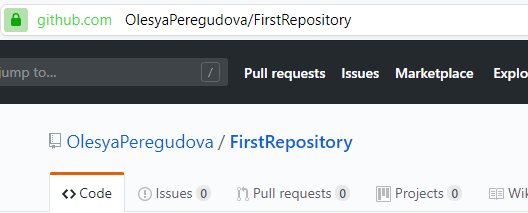
**Задачи:** 1. Регистрация на GitHub.

2. Установка клиента SmartGit

3. Описание основных понятий и команд Git

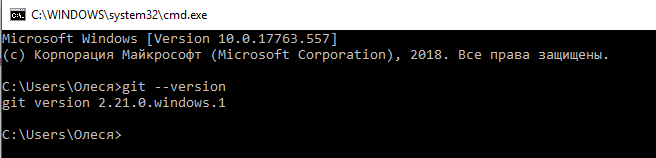
4. Практическая часть.Выполнение заданий.

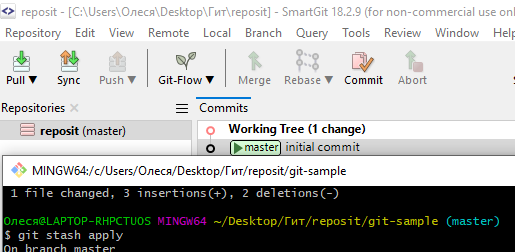
**1. Регистрация на GitHub.**



**2.** **Установка клиента SmartGit.**

Установка клиента SmartGit прошла успешно.





**3. Описание основных понятий и команд Git.**

*Основные понятия*

**Git** — система контроля и управления версиями файлов.

**GitHub** — веб-сервис для размещения репозиториев и совместной разработки проектов.

**Репозиторий Git** — каталог файловой системы, в котором находятся: файлы конфигурации, файлы журналов операций, выполняемых над репозиторием, индекс расположения файлов и хранилище, содержащее сами контролируемые файлы.

**Локальный репозиторий** — репозиторий, расположенный на локальном компьютере разработчика в каталоге. Именно в нём происходит разработка и фиксация изменений, которые отправляются на удалённый репозиторий.

**Удалённый репозиторий** — репозиторий, находящийся на удалённом сервере. Это общий репозиторий, в который приходят все изменения и из которого забираются все обновления.

*Основные команды Git*

Команда **git add** добавляет содержимое рабочей директории в индекс (staging area) для последующего коммита. По умолчанию git commit использует лишь этот индекс.

Команда **git status** показывает состояния файлов в рабочей директории 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.

Команда **git diff** используется для вычисления разницы между любыми двумя Git деревьями. Это может быть разница между рабочей директорией и индексом (собственно git diff), разница между индексом и последним коммитом (git diff --staged), или между любыми двумя коммитами (git diff master branchB).

Команда **git commit** берёт все данные, добавленные в индекс с помощью git add, и сохраняет их слепок во внутренней базе данных, а затем сдвигает указатель текущей ветки на этот слепок.

Команда **git reset,** как можно догадаться из названия, используется в основном для отмены изменений. Она изменяет указатель HEAD и, опционально, состояние индекса. Также эта команда может изменить файлы в рабочей директории при использовании параметра --hard, что может привести к потере наработок при неправильном использовании, так что убедитесь в серьёзности своих намерений прежде чем использовать его.

Команда **git rm**используется в Git для удаления файлов из индекса и рабочей директории. Она похожа на git add с тем лишь исключением, что она удаляет, а не добавляет файлы для следующего коммита.

Команда **git mv** — это всего лишь удобный способ переместить файл, а затем выполнить git add для нового файла и git rm для старого.

Команда **git clean** используется для удаления мусора из рабочей директории. Это могут быть результаты сборки проекта или файлы конфликтов слияний.

*Для выполнения ветвления и слияния*

Команда **git branch** — это своего рода “менеджер веток”. Она умеет перечислять ваши ветки, создавать новые, удалять и переименовывать их.

Команда **git checkout** используется для переключения веток и выгрузки их содержимого в рабочую директорию.

Команда **git merge** используется для слияния одной или нескольких веток в текущую. Затем она устанавливает указатель текущей ветки на результирующий коммит.

Команда **git log** используется для просмотра истории коммитов, начиная с самого свежего и уходя к истокам проекта. По умолчанию, она показывает лишь историю текущей ветки, но может быть настроена на вывод истории других, даже нескольких сразу, веток. Также её можно использовать для просмотра различий между ветками на уровне коммитов.

Команда **git stash** используется для временного сохранения всех незакоммиченных изменений для очистки рабочей директории без необходимости коммитить незавершённую работу в новую ветку.

Команда **git tag** используется для задания постоянной метки на какой-либо момент в истории проекта. Обычно она используется для релизов.

*Для работы с удаленными репозиториями:*

Команда **git fetch** связывается с удалённым репозиторием и забирает из него все изменения, которых у вас пока нет и сохраняет их локально.

Команда **git pull** работает как комбинация команд git fetch и git merge, т.е. Git вначале забирает изменения из указанного удалённого репозитория, а затем пытается слить их с текущей веткой.

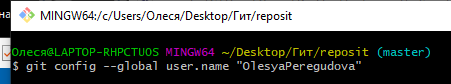
Команда **git push** используется для установления связи с удалённым репозиторием, вычисления локальных изменений отсутствующих в нём, и собственно их передачи в вышеупомянутый репозиторий. Этой команде нужно право на запись в репозиторий, поэтому она использует аутентификацию.

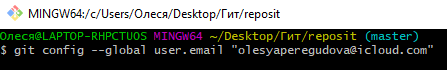
Команда **git remote** служит для управления списком удалённых репозиториев. Она позволяет сохранять длинные URL репозиториев в виде понятных коротких строк, например «origin», так что вам не придётся забивать голову всякой ерундой и набирать её каждый раз для связи с сервером. Вы можете использовать несколько удалённых репозиториев для работы и git remote поможет добавлять, изменять и удалять их.

**4. Практическая часть. Выполнение заданий.**

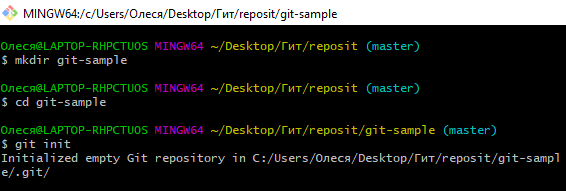
**Задание 1**. Первоначальная настройка системы. Создание конфигурации.

Используя команду GIT CONFIG – GLOBAL, создаем глобальную конфигурацию. Для этого вводим имя и e-mail пользователя.





Создаем папку git-sample, в которой в дальнейшем будут храниться наши файлы.



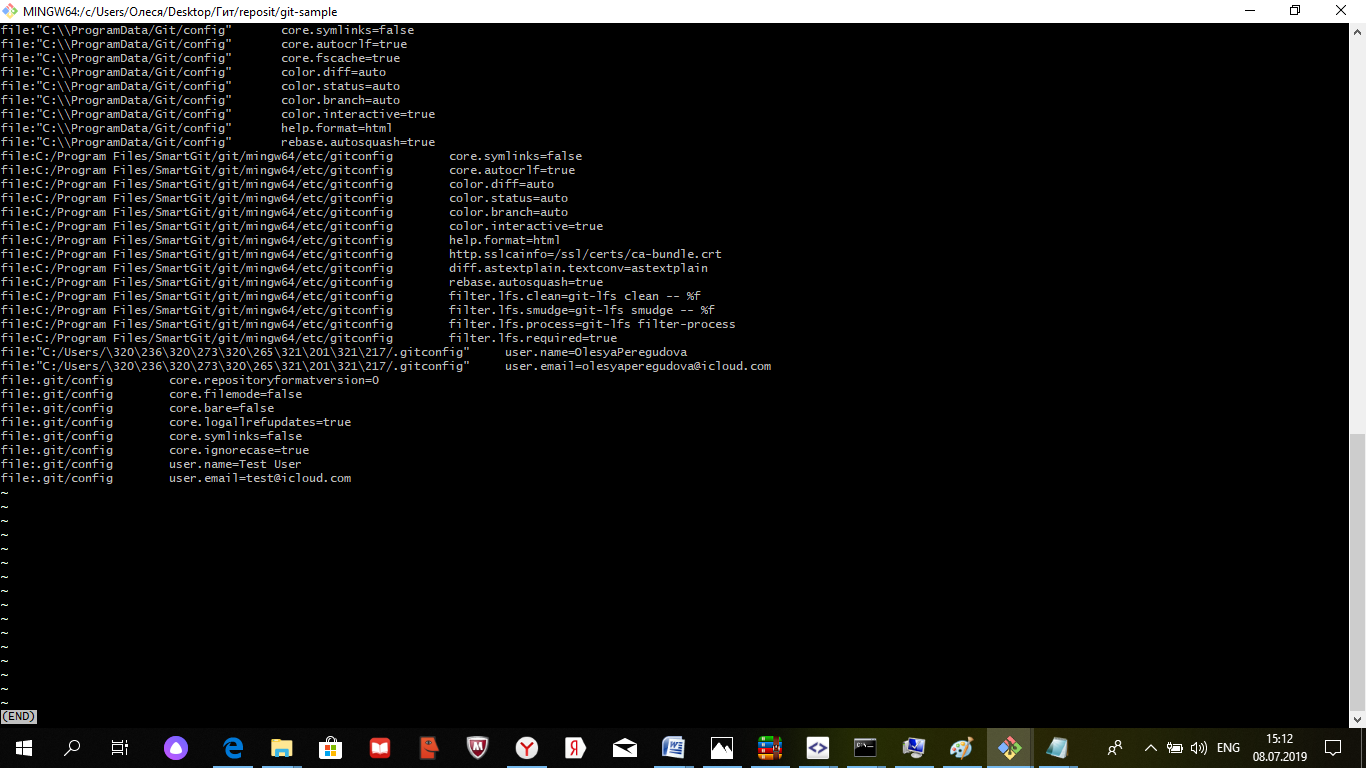
С помощью команд LS и CD мы можем узнать,что находится в папке.



Создаем локальную конфигурацию с именем пользователя «Test» и e-mail «test@icloud.com»



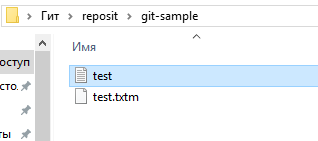
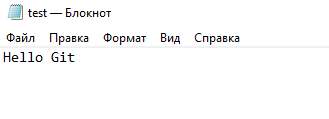
Командой *GIT CONFIG --LIST --SHOW-ORIGIN*  проверяем наличие локальной конфигурации



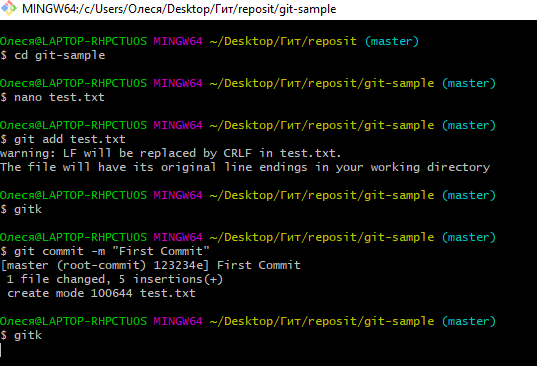
Конфигурация создана успешно.

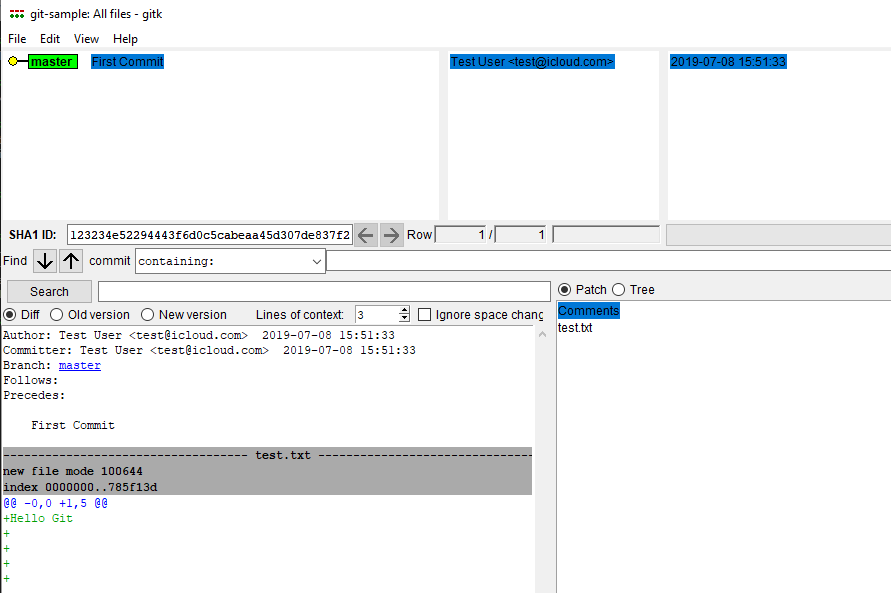
**Задание 2**. Создать файл, изменить его, сделать коммит.

Создадим файл «Test» используя NANO.



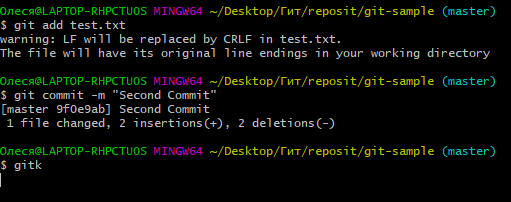
Используя команду GIT COMMIT сохраним наши изменения.





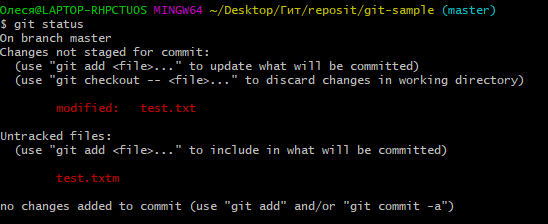
Далее изменяем файл и делаем второй коммит.



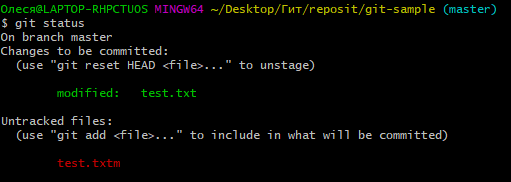


**3 задание.** Проверка состояния git-репозитория c помощью команды git status:

1. Выполнили команду git status до того как добавили наш файл в кандидаты на коммит.



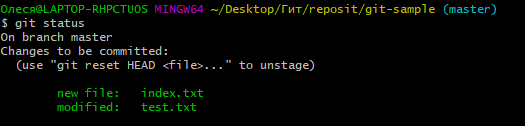
1. После добавления файла в кандидаты на коммит.

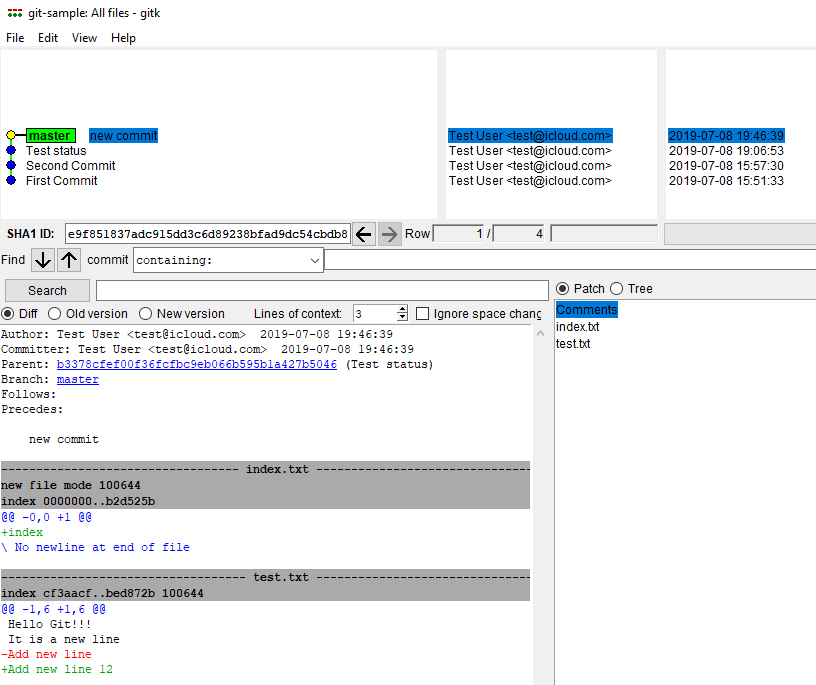


Данная команда позволяет посмотреть статус любого файла и наиболее часто применяется, когда необходимо узнать какие файлы были изменены из большого количества файлов.

**4 задание.** Индексация изменений.

Удалим один из файлов, а затем с помощью команд git add \*, git status и git commit узнаем какой из файлов был удален, а какой изменен.



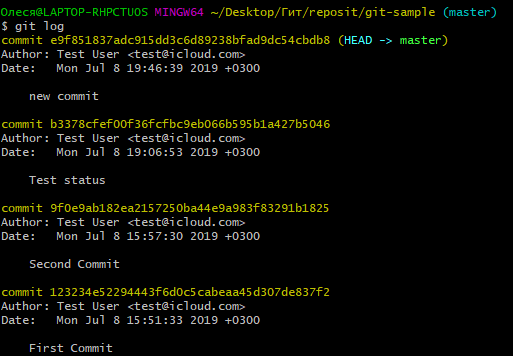


Изменения файлов были проиндексированы. Это означает, что git теперь знает об изменении, но изменение пока не записано в репозиторий. Следующий коммит будет включать в себя проиндексированные изменения.

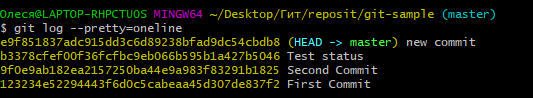
Таким образом, Git , индексируя новые изменения, дает нам понять, что произошло с данным файлом.

**5 задание**. История изменений

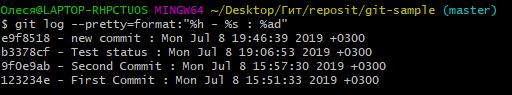
Используя команду GIT LOG, выведем историю коммитов на консоль:



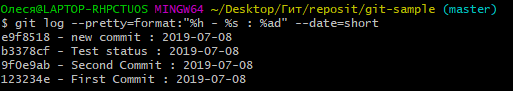
Выведем каждый коммит в одну строку, используя флаг pretty:



Изменим формат вывода коммитов с помощью флага format:

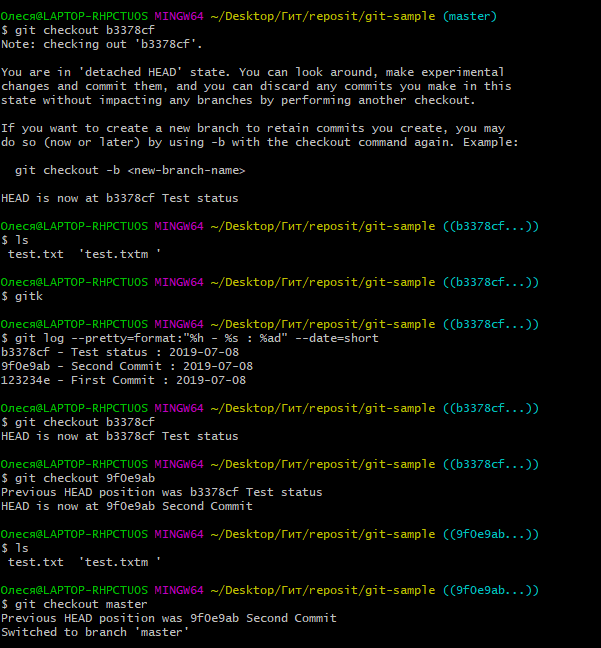


Изменим формат вывода даты коммита с помощью флага date:



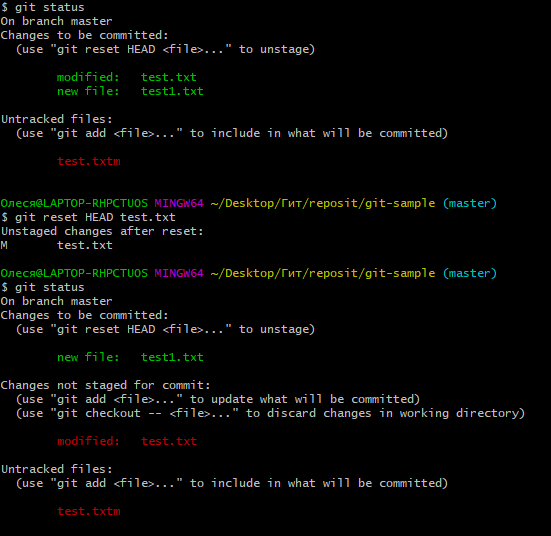
**6 задание.** Использование команды GIT CHECKOUT

Данная команда чаще всего используется для поиска ошибок при командной работе. Она необходима, когда нужно вернуться в состояние определенного коммита.



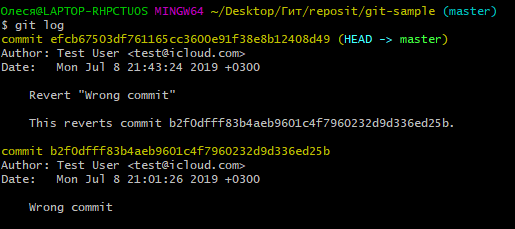
**7 задание.** Отмена изменений

В случае если мы захотим отменить изменения, которые мы внесли в файл, используется команда GIT RESET.



**8 задание.** Удаление коммитов.

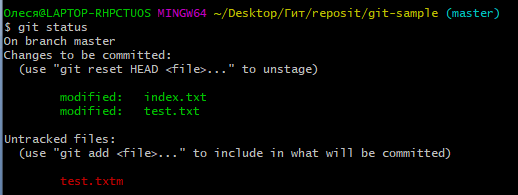
Для того, чтобы удалить коммит, необходимо использовать команду GIT REVERT –NO-EDIT

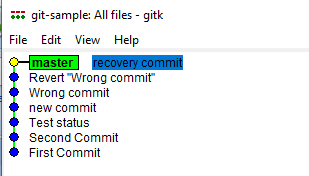


**9 задание.** Решение конфликта выбора действий

После удаления коммита, показанного в примере выше, часто стоит задача решения конфликта выбора последующих действий над файлами.

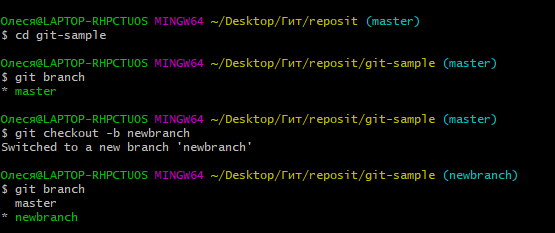
Для решения данной задачи используется команда GIT REVERT –CONTINUE. Изменяем данные в файлах, добавляем файлы в кандилаты на коммит, после чего проверяем git status и коммитим данное состояние ( Recovery Commit)





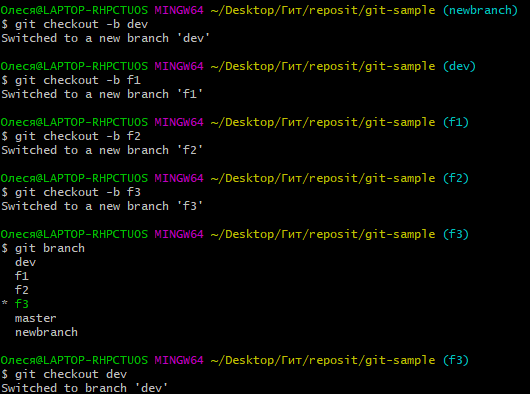
**10 задание.** Использование ветвей.

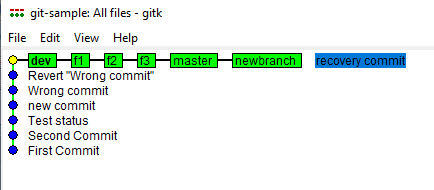
Создание ветвей



Для того чтобы создать ветвь используется команда GIT CHECKOUT -B <name\_of\_branch>

Чтобы просмотреть все существующие ветви используется git branch,а чтобы переключиться с одной ветви на другую GIT CHECKOUT < name\_of\_branch >

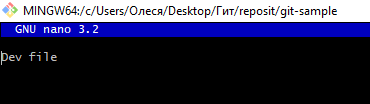




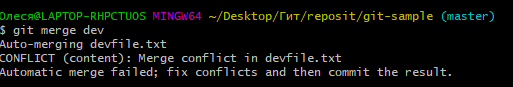
**11 задание**. Слияние с использованием GIT MERGE .

Данная команда используется для выполнения слияния двух веток .

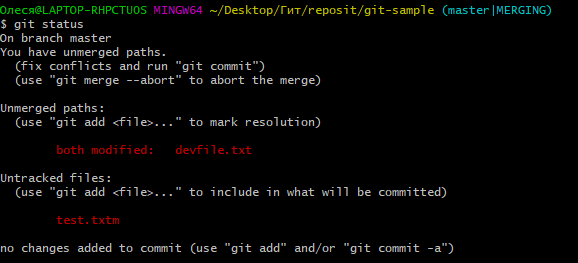
Сначала изменим файл devfile.txt в обеих ветвях.



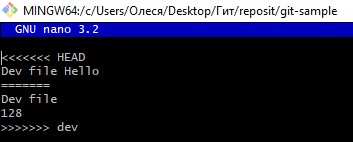
Выполним слияние двух ветвей.



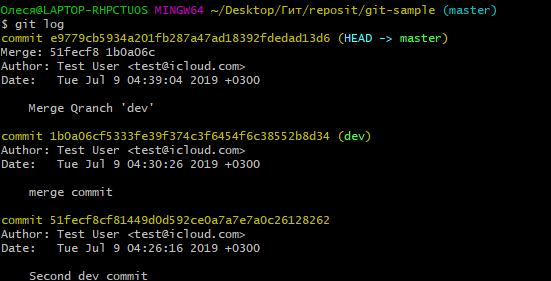
Проверим git status. Видим, что обнаружен конфликт.



Для решения конфликта необходимо в ветви Master открыть файл и вручную изменить его, убрав лишнее.

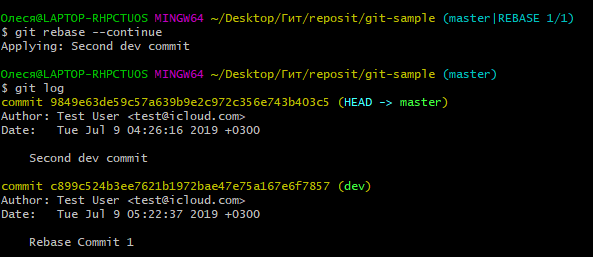


После чего выполняем коммит, и он сохраняется в истории с собственным комментарием.



**12 задание**. Использование команды GIT REBASE

Изменив файлы и выполнив коммит, используем команду REBASE чтобы переместить коммиты с одной ветви на другую.

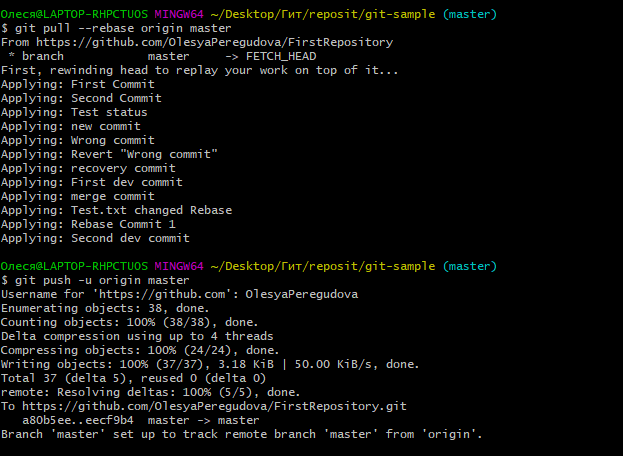


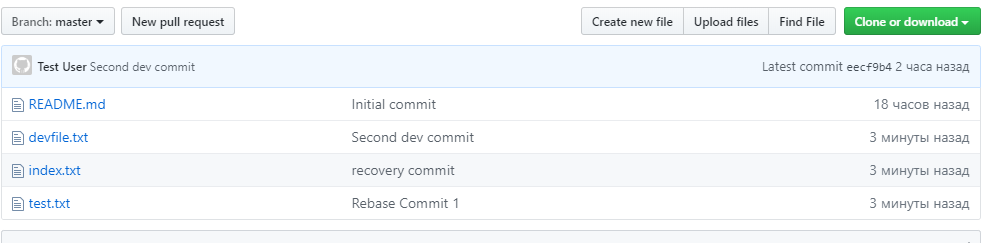
Интересно заметить, что REBASE в отличие от MERGE не сохраняет исорию коммитов.

**13 задание.** Работа с удаленным репозиторием.

Загрузим наши коммиты на удаленный репозиторий на GitHub, используя следующие команды:

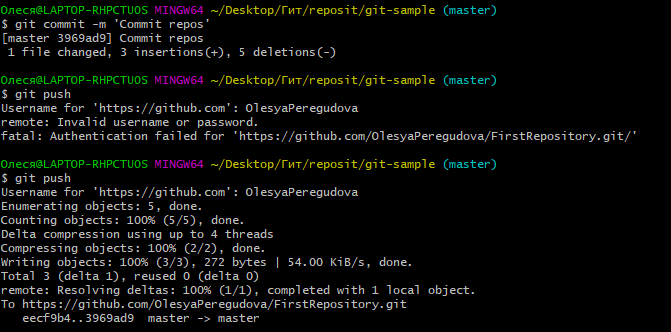


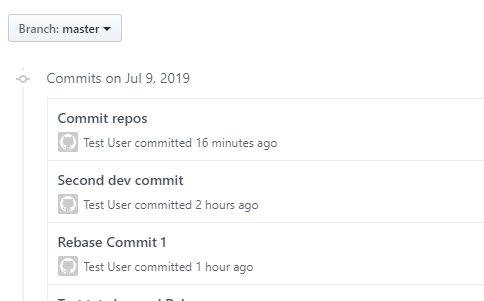




**14 задание**. Использование команды GIT PUSH

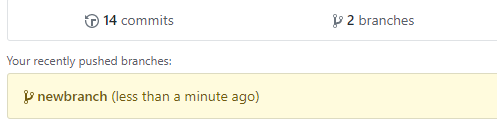
С помощью данной команды мы можем загружать на удаленный репозиторий новые изменения.





Также мы можем добавить новую ветвь и, используя Git Push, залить ее на удаленный репозиторий.

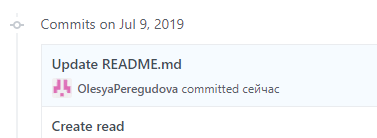


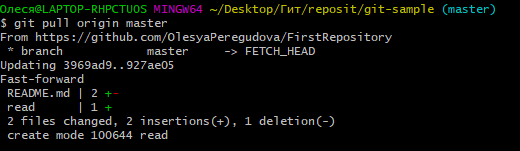


**Задание 15.** Использование команды GIT PULL.

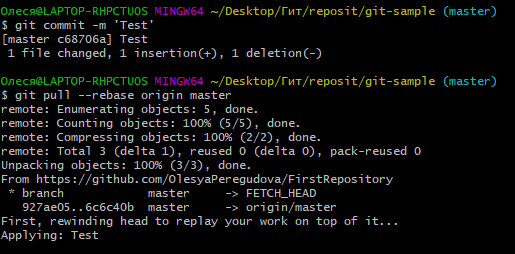
Данная команда позволяет «достать» изменения из удаленного репозитория в локальный.

В данном задании мы изменили файл в удаленном репозитории на сервисе GitHub и выполнили команду GIT PULL.

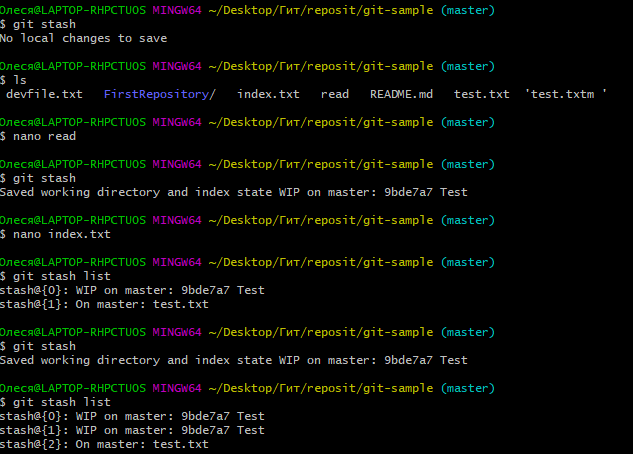




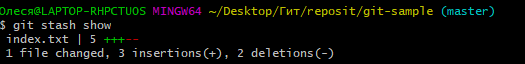
Также, используя команду GIT PULL –REBASE, мы вытянули все удаленные изменения и при этом избежали конфликтов в коде.



**Задание 16.** Использование GIT STASH..



Чтобы показать последнее изменение добавленное в STASH используется GIT STASH SHOW



Чтобы очистить STASH используют GIT STASH CLEAR.

