<https://habr.com/ru/companies/yandex_praktikum/articles/769644/>

«спулил изменения» значит «забрал из удалённого репозитория изменения, сделанные другими разработчиками».

«Запушил фикс» — отправил исправления в удалённый репозиторий.

«Поймал конфликт» — столкнулся с пересечениями в изменениях кода.

на GitHub любой человек может создать репозиторий и хранить там своё портфолио или базу знаний. Например, вы создавали тест-кейсы и хотели бы их приложить к резюме — выложите их на GitHub и дайте работодателю ссылку на репозиторий. Это здорово продемонстрирует ваши скилы (навыки).

Установка git дистр

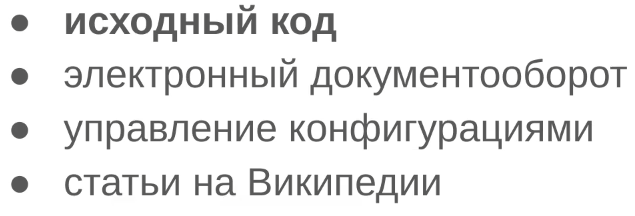
<https://git-scm.com/book/ru/v2/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%A3%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B0-Git>

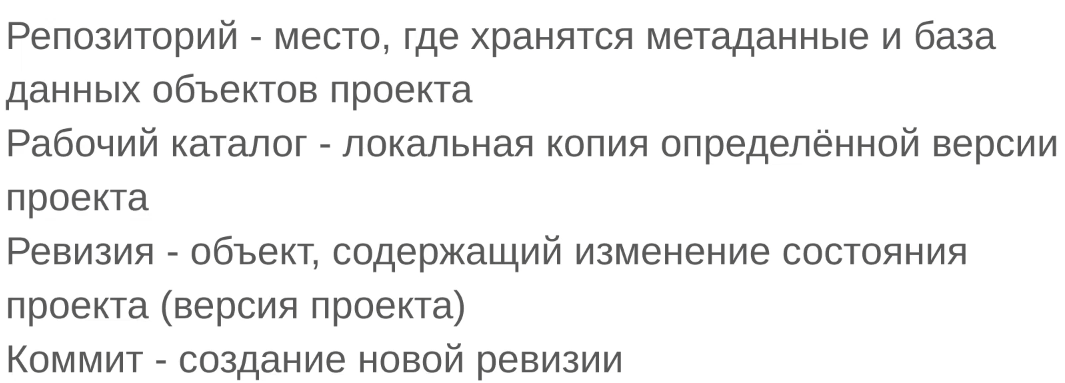
**Мануал установка настройка**

https://selectel.ru/blog/tutorials/how-to-install-git-to-windows/

Git – распределенная система контроля версий.

Хранят





**Что такое SSH-клиент?**

Клиент SSH обеспечивает безопасную среду для подключения к удаленному компьютеру . Аутентификация завершается с использованием имени пользователя и пароля. Если вам требуется доступ к командной строке без ввода пароля, для аутентификации используются принципы открытого и закрытого ключей.

**Первоначальная настройка и создание репозитория**

Перед началом работы с Git нужно указать свое имя и email, которые в дальнейшем будут записываться в историю изменений при каждом коммите. В будущем это позволит понять, кто именно внес те или иные изменения.

Откроем любое из приложений — Git Bash или Git CMD. Первое — это командная строка в стиле Linux, второе — командная строка в стиле Windows. Выбирайте то, что вам ближе. Мы выберем Git Bash и выполним две команды:

**git config --global user.email "git-user@selectel.ru"**

**git config --global user.name "Selectel Git User"**

Теперь Git полностью готов к работе. Давайте создадим репозиторий и зафиксируем в нем первое изменение (сделаем коммит). Для начала создадим каталог для будущего репозитория и сразу перейдем в него:

**mkdir first-repo && cd first-repo**

**mkdir** *создать новую директорию*.

**Cd** *текущая рабочая директория изменяется на указанную.*

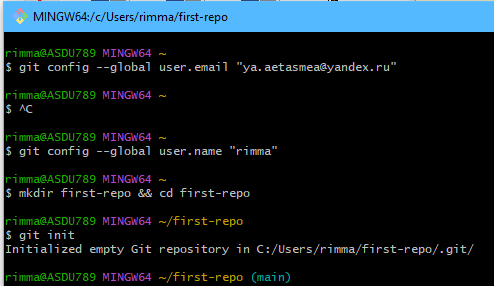
Создаем новый репозиторий в этом каталоге:

*Инициализация выполняется в принципе один раз на проект, когда вы создаете что-то новое. После этого забываем про git init.*

**git init**

Увидим ответ:

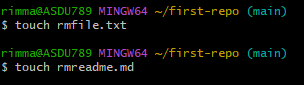
**Initialized empty Git repository in C:/Users/git\_user/first-repo/.git/.**



Это означает, что в директории создан новый репозиторий. **Далее создадим текстовый файл**, назовем его README.md, и напишем в нем любой текст. Но сам по себе этот файл не попадет в следующий коммит. Мы должны проиндексировать изменения, то есть явно сказать Git-у, что этот файл нужно учитывать в следующем коммите:

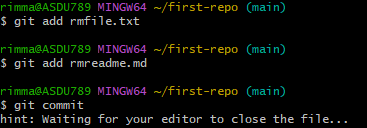
**touch rmfile.txt создали файл**

**git add rmfile.txt добавляем файлы в гит чтобы они отслеживались**

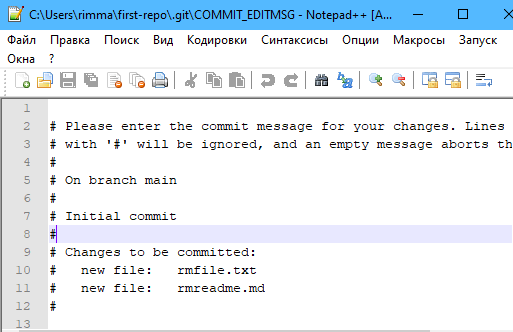


Далее введем команду:

**git commit**



Откроется текстовый редактор, который мы выбирали на этапе установки Git. Тут нам нужно ввести комментарий для коммита, то есть кратко описать изменение, которое мы сделали. Мы напишем такой комментарий:





Обратите внимание, что Git автоматически добавил подсказку в это окно. При этом она не войдет в коммит, потому что в начале строки стоит символ решетки, и Git проигнорирует ее. Но она может быть полезна для дополнительной проверки: мы видим название текущей ветки и список файлов, которые войдут в коммит.

Сохраним файл и закроем редактор. Увидим примерно следующее сообщение:



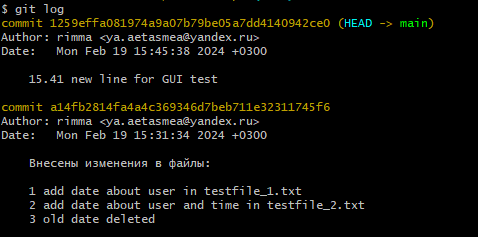
Мы успешно сделали первый коммит.

Сообщение содержит следующие данные о коммите:

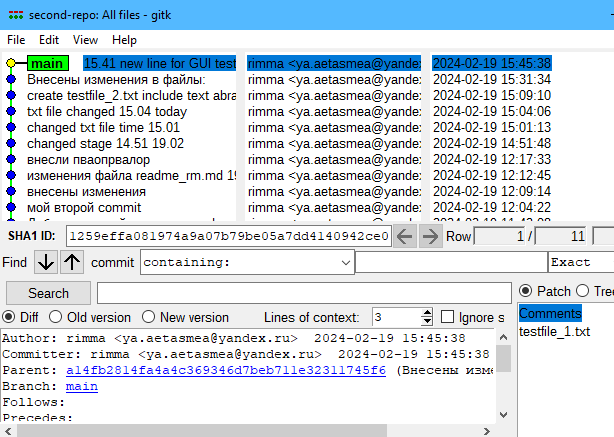
* **master** — для какой ветки он был выполнен
* **root-commit**— является корневым (первым) коммитом
* **1e36e0f** — уникальный идентификатор (контрольная сумма)
* **Инициализация проекта** — заголовок описания к коммиту. При этом другие пункты — не отображаются
* **2 files changed**— количество измененных файлов
* **6 insertions(+)** — количество добавленных в файлах строк
* **create mode 100644** — права доступа к файлам в [Unix-стиле](https://youtu.be/KrlYu1ToS-o" \t "_blank)

Первая команда, которую вы скорей всего будете выполнять чаще всего **git status**. 

**git log** информация по сделанным коммитам



**gitk&** запуск инструмента для отслеживания ветвей



**GIT для тестировщика**

**Для git cmd**

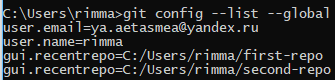
Команда git команды инфа . убедиться что встал

copy NUL .gitconfig





git config --list –global проверяем инфу в файле .gitconfig



**1 Регистрация на github.com**

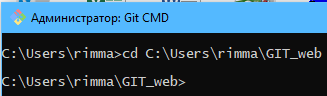
Git это клиентсерверная архитектура. Клиент git это то , что установлено на мой компьютер. Сервер внутренний репозиторий компании, где хранится код или github с публичными репозиториями кода для разработчиков на аутсорсе.

<https://github.com/rimma-er/Rm_storage>

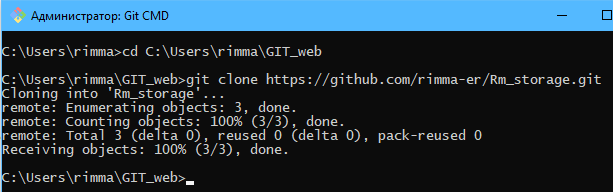
**2 Копируем репозиторий из github на локальный компьютер в подготовленную папку.**

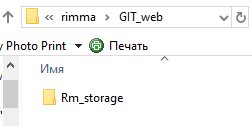
Создать папку для хранения репозитория C:\Users\rimma\**GIT\_web**

Перейти в папку **GIT\_web**



Копируем репозиторий в локальную папку **git clone https://github.com/rimma-er/Rm\_storage.git**

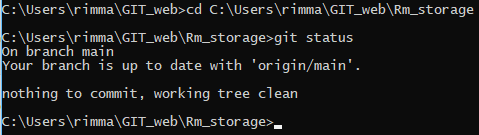




**3 Локальная работа с репозиторием из git hub**

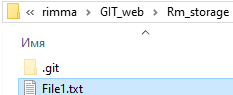
Преходим в локальный репозиторий C:\Users\rimma\GIT\_web\**Rm\_storage**

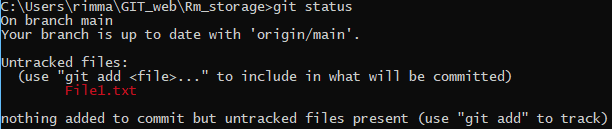
Смотрим его текущее состояние. Какие есть действия /данные.



Создать файл File1.txt в папке **Rm\_storage**

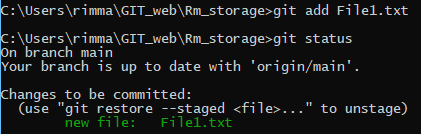
Смотрим git status. Файл не отслежен (untracked)



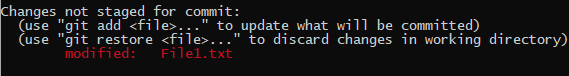


Сделать запись в файл. Показать файл гиту. git add File1.txt

Проверить состояние git status. Теперь он в системе.

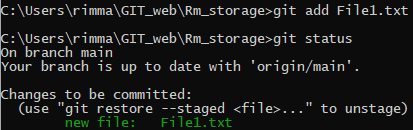


Изменим текст в файле и git status. Изменения система зафиксирует теперь.



Можно **откатиться к начальной** версии или **согласиться с изменениями**.

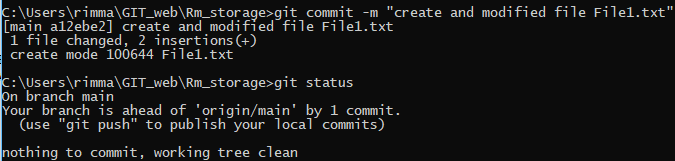
Соглашаемся. git add File1.txt git status. Остается только конечная верссия.



**4 Добавляем файл в git hub в браузере**

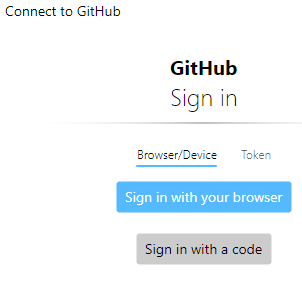
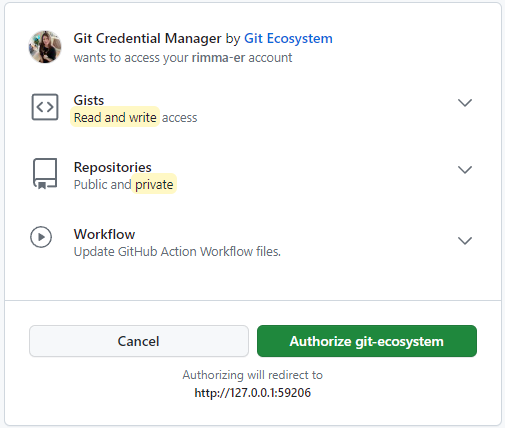
Фиксируем изменения с подробным описанием что сделали. Это локально.

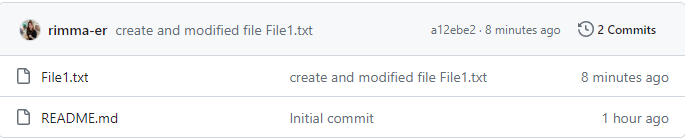
**git commit -m "create and modified file File1.txt"** и **git status**



Чтобы отправить файл на удаленный репозиторий git hub **git push**





**5 Добавляем файл с изменениями в локальный репозиторий (синхронизация)**

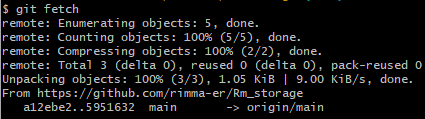
Изменили файл в git hub

В локальной папке открываем git bush

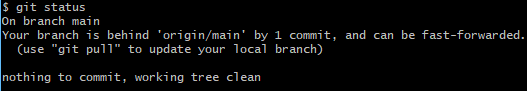
git status

cat File1.txt смотрим содержимое файла. Пока не видит локально изменения



git fetch показывает что есть изменения в вебе, ничего не меняет 

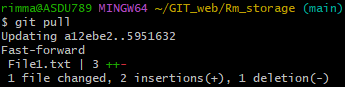
**git status** показывает инфу что локальная ветка отстает на один коммит.

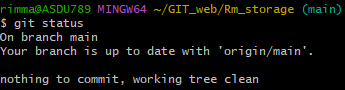


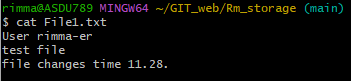
**git pull** вносит изменения из удаленного репозитория в локальный. Ветки синхронизируются

**git status**

**cat File1.txt** видим что новые изменения содержимого появились





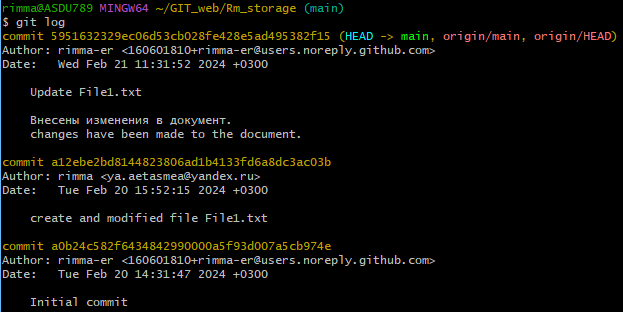


Локальная и удаленная директории синхронизированы.

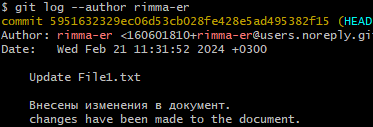
**Updating a12ebe2..5951632** этот номер - хэш коммита, по нему можно отследить все комиты которые проводились



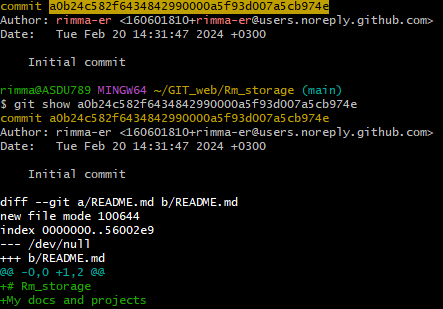
**git log** смотрим все проделанные комиты.



**git log --author rimma** См изменения определенных авторов



**git show** номер хэша См содержание определенного комита



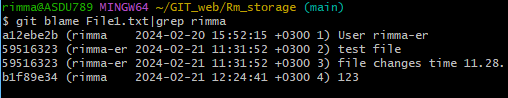
**git show** инфа по самому последнему комиту

**git blame File1.txt** инфа по автору комита, какие изменения внес

**git blame File1.txt|grep 123** Кто вносил изменения по конкретной строке в файле



**git blame File1.txt|grep rimma** какие строки вносил конкретный автор в этот файл.



**echo "add line" >> File1.txt** добавить строку в файл (добавили строку с текстом add line)

**git diff** см разницу между версиями файла после изменений

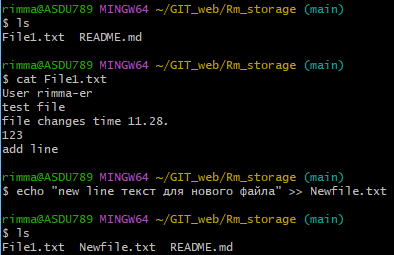
Синхронизируем файлы локальный и удаленный **git commit –m** и **Git push**



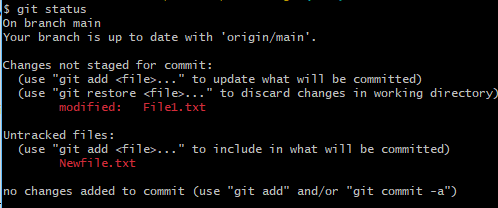
**Простые конфликты и откаты изменений в GIT.**

Добавляем файл в локальную папку c текстом внутри

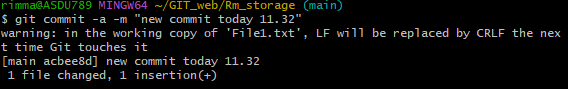
**echo "new line текст для нового файла" >> Newfile.txt**



Git status



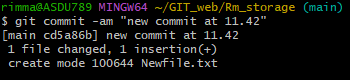
Добавляем все файлы (a) и сообщение для коммита (m) **git commit -a -m "new commit today 11.32"**



Откат изменений последнего коммита git reset Head~1

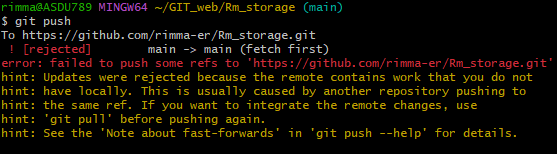


Git add Newfile.txt commit

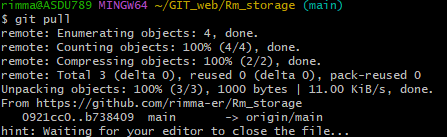


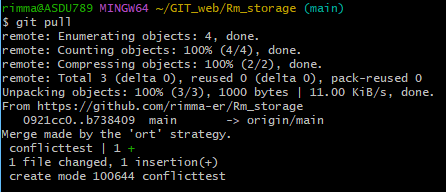
Для **конфликта (когда сеть и локалка не синхронизированы между собой)** создаем файл conflicttest на гит хаб

И делаем git push , получаем конфликт

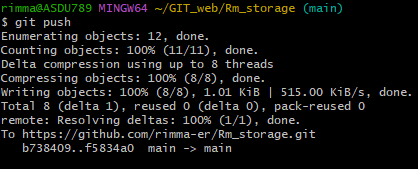


Надо залить изменения из гит хаб в локальный репозиторий



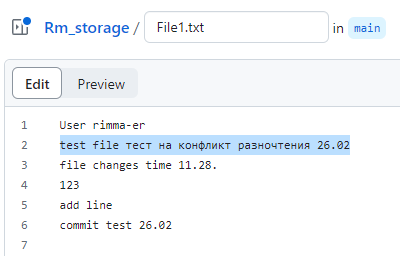
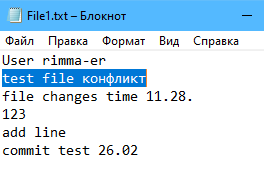


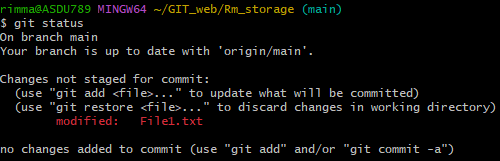
Снова залить изменения в гит хаб из локальн репозитория

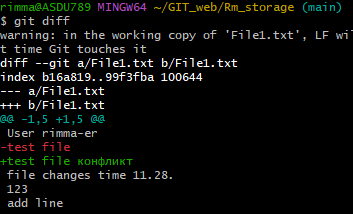


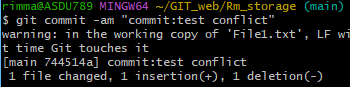
Обновить гит хаб. Проверить что все есть.

**конфликт разночтения (когда в одном и том же файле в сети и локалке отличается строка текста)**

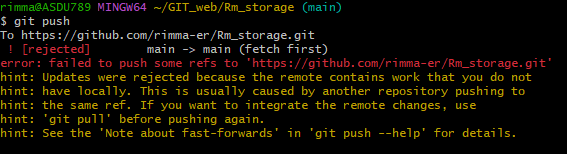
 

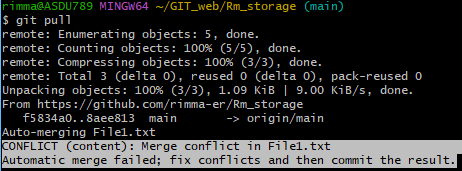




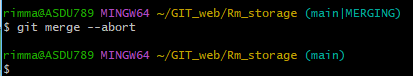


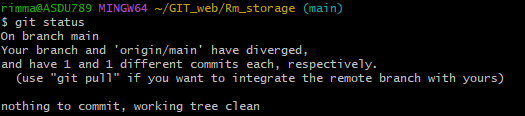
При попытке залить в гит хаб и локалку изменения будет конфликт





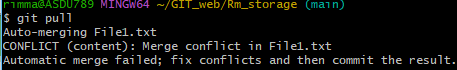
Решение 1. Если мы не знаем в чем разногласие информации обратиться к разработчику чтобы он решал вопрос.

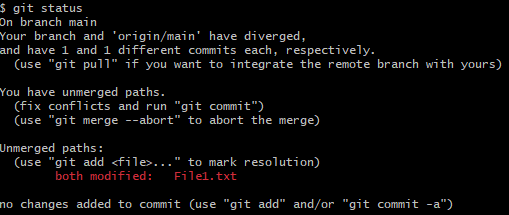




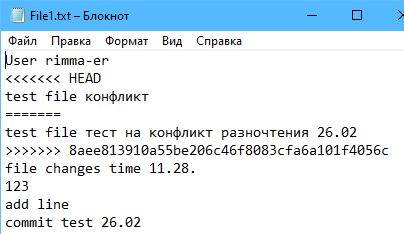
Решение 2.

Снова воспроизводим проблему

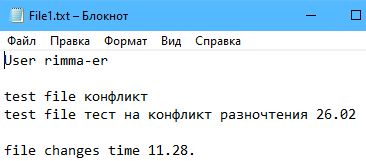




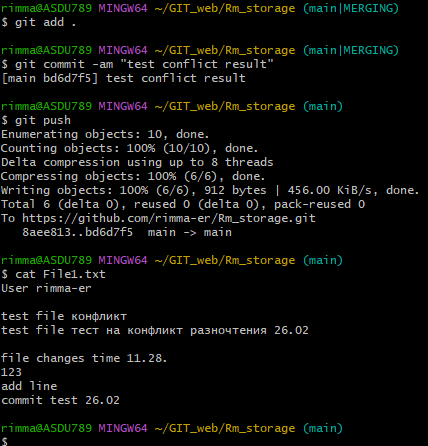
Система сама указывает где конфликт



Убираем лишнее. Оставляем только нужные строки.



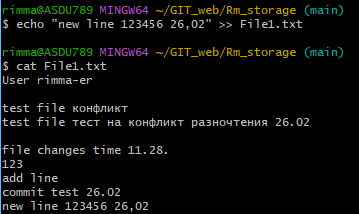
Далее



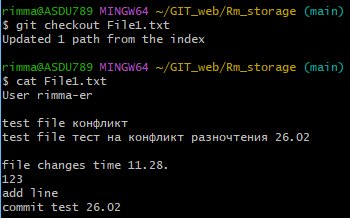
**git add .** Относительный путь . обозначает текущую директорию, т.е. ту, в которой была запущена команда. Таким образом, если вы выполняете команду add не в корневой директории проекта (той, где лежит .git/), то будет обработано содержимое только текущей директории.

**Откат изменений checkout**

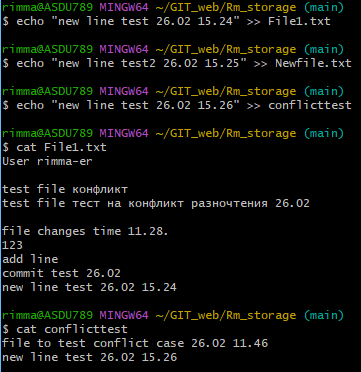
Добавим строку с файл



Откатываем изменение для одного файла. Чтобы строки не было



Добавим строки в несколько файлов



Откатываем изменение для нескольких файлов

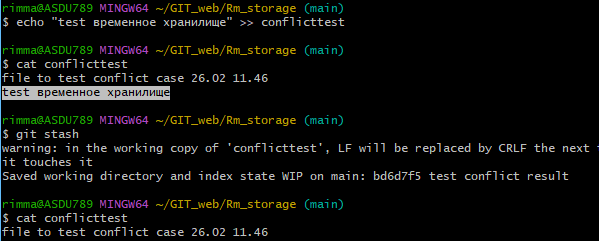


**Добавить изменения во временное хранилище:**

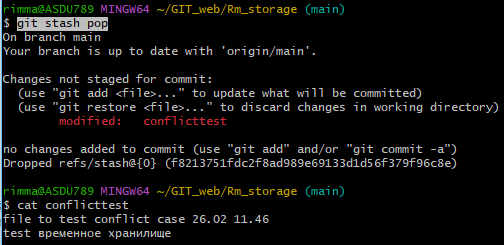
Добавляем строку.

git stash помещаем во временное хранилище

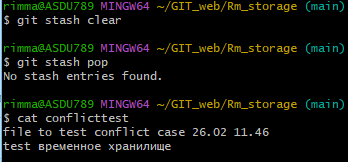
строки больше не видно в файле



Вернуться к изменениям

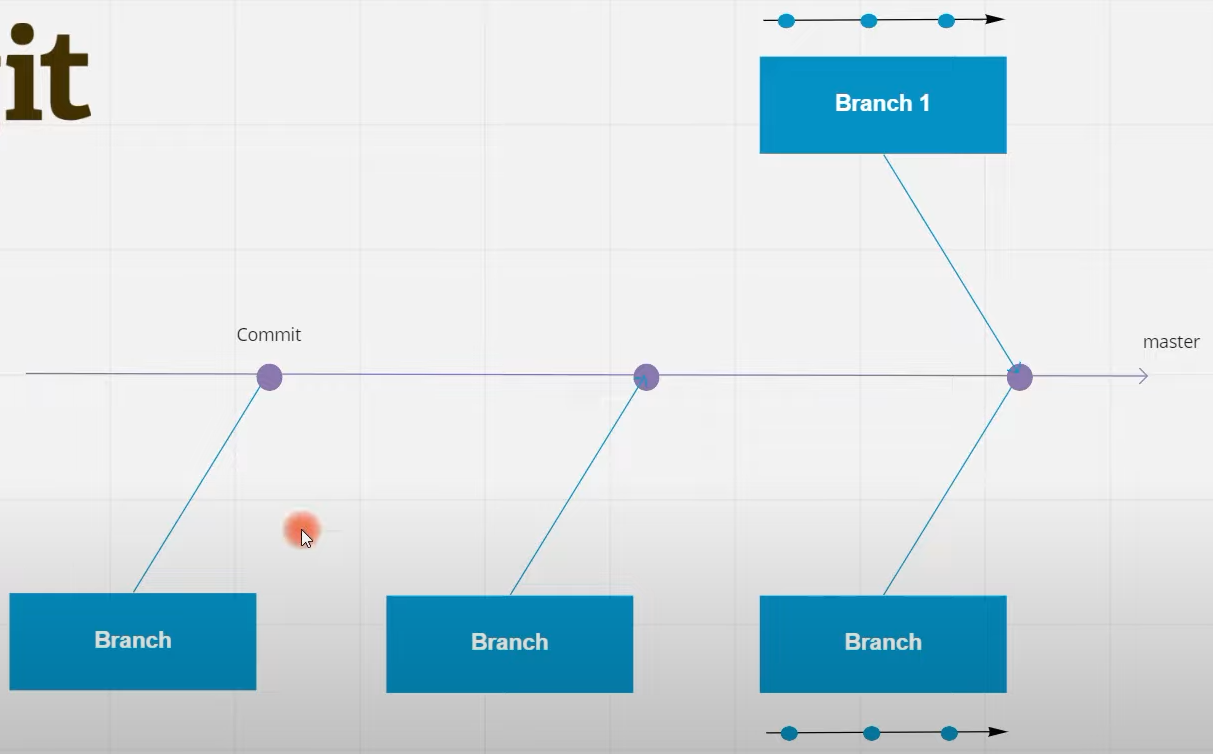


Удалить строку навсегда из временного хранилища. В файле строка останется



**Git stash** перемещает текущие изменения (так называемые local changes) в локальную директорию, которая выполняет роль специального хранилища, то есть скрывает эти изменения, сохраняя их отдельно, с опцией вернуть позже, когда это понадобится. Все «прятанья», создаваемые git stash, делятся на две категории: **индексированные** и **неиндексированные**. С помощью откладываемых изменений можно удобно и оперативно работать в нескольких ветках, не рискуя потерять важные участки кода. можно избежать некоторых сложностей в процессе разработки мобильных приложений, веб-приложений и других продуктов.

**Создание дополнительной ветки**



Сделать корневым каталог **cd Rm\_storage**

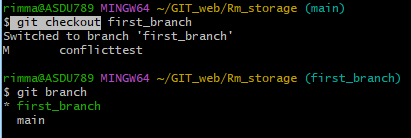
Создать ветку git branch first\_branch

Проверить что она есть (основная ветка помечена \*)

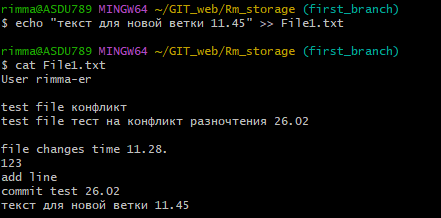
****

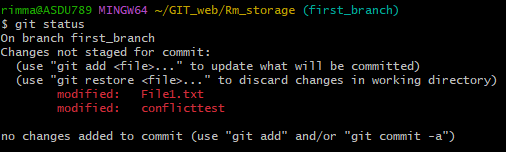
Переключиться на новую ветку с названием first\_branch

Проверить что переключились

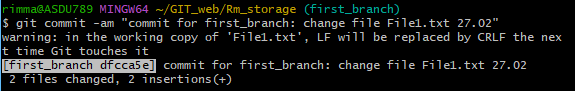


Добавить текст в файл. Комиты по изменениям будут в новой ветке.





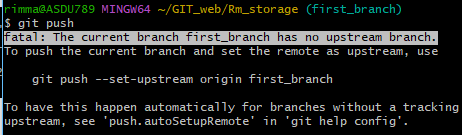
Комит по изменениям для новой ветки



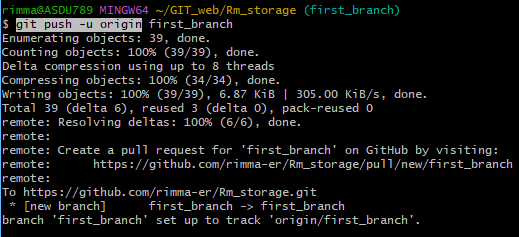
Сохраняем изменения в гит хаб

Получаем ошибку фатальный: текущая ветка first\_branch не имеет восходящей ветки.

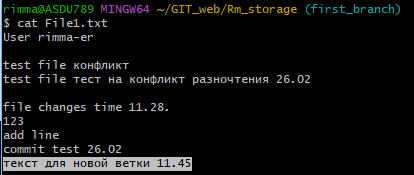
Тк мы не создавали ветку first\_branch в гит хаб



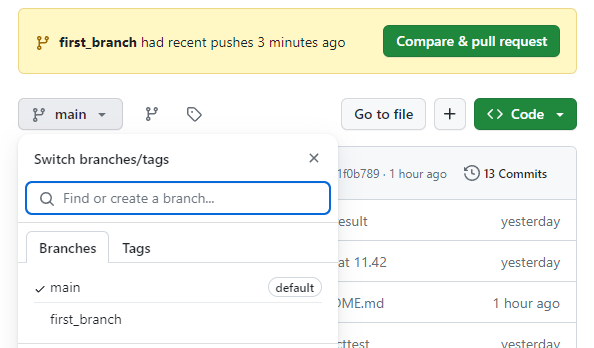
Связываем локальную ветку с удаленной

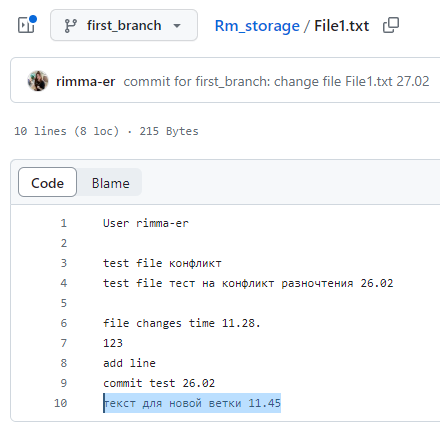


Проверяем как выглядит измененный файл



Проверяем есть ли ветка на гит хабе





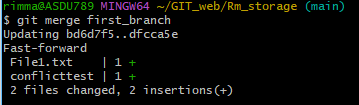
Закончили вносить изменения в новую ветку и хотим залить изменения в основную ветку main. Нужен merge



Для этого Вернуться в основную ветку.



Заливаем новую ветку в основную.

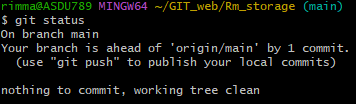


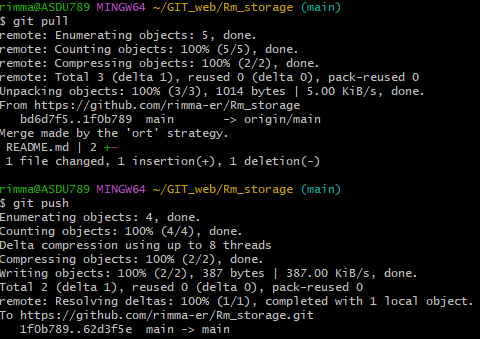
Updating bd6d7f5 последний коммит в текущей ветке

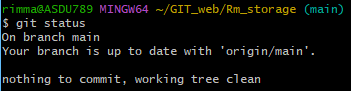
.dfcca5e хэш коммита который последним добавлен в основную ветку.

Fast-forward стратегия по которой происходило слияние (есть еще no Fast-forward)

Можем посмотреть статус и увидеть что есть один коммит который надо сделать.







**Конфликты во время merge (Слияния)**

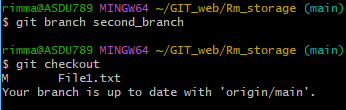
Пусть локальный и удаленный репозиторий отличаются.

Завести файл [conflict\_branch](https://github.com/rimma-er/Rm_storage/blob/main/conflict_branch) на гит хабе

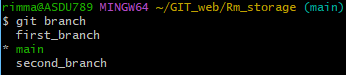
File1.txt изменить в локалке.

Добавляем новую ветку

Чеком проверяем что получилось / файл изменен



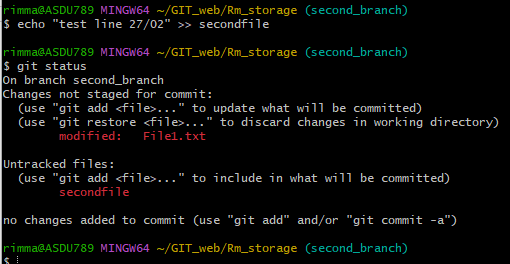
Проверяем ветки

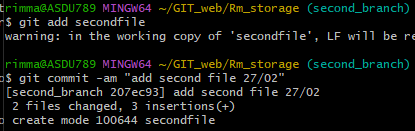


Переключиться на вторую ветку

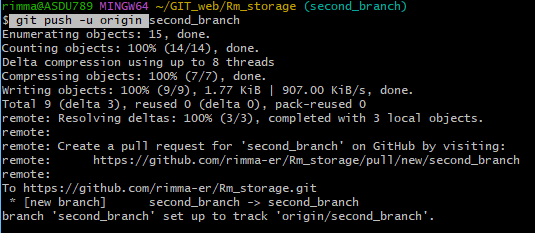


Создаем новый файл для второй ветки





Связываем локальную ветку с удаленной (локальный репозиторий)

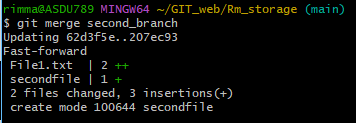


Делаем слияние основной ветки и второй (локальный репозиторий)

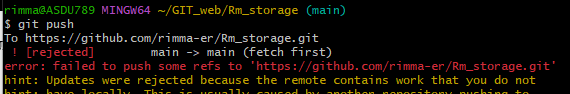
Для этого переключиться на основную ветку

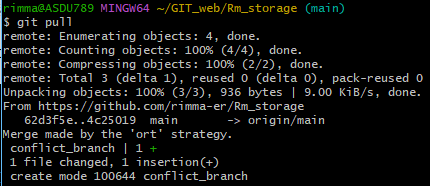


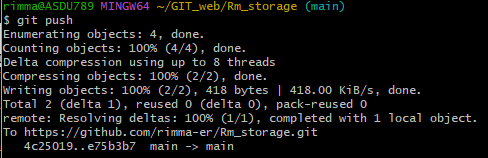
Делаем слияние второй ветки с первой (локальный репозиторий)



Заливаем в гит хаб. Будет ошибка. Несоответствие данных



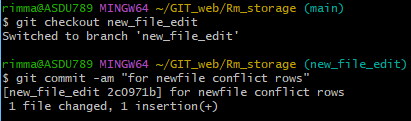




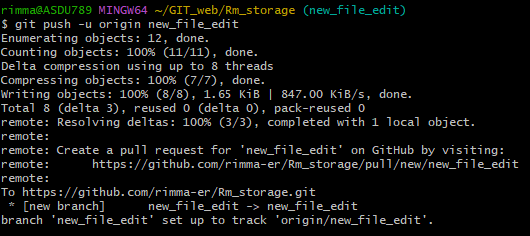
Ошибка при изменении одного и того же файла , одной и той же его строки



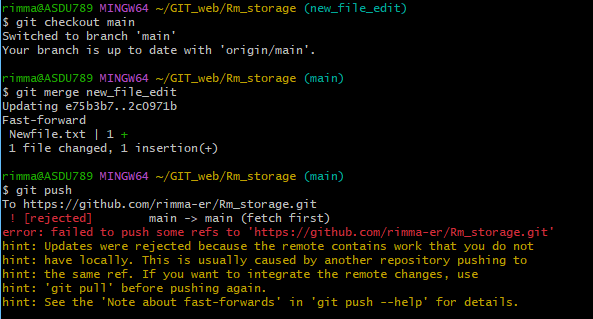
Меняем содержание строк файла локально и в гит хаб



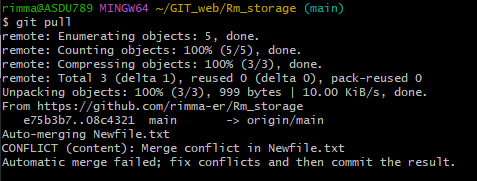
локально



Слияние и отправка в гит хаб

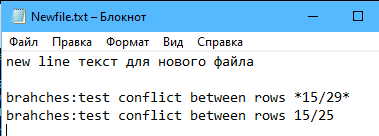


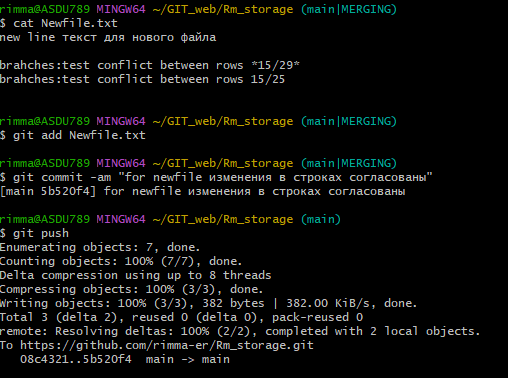
Конфликт

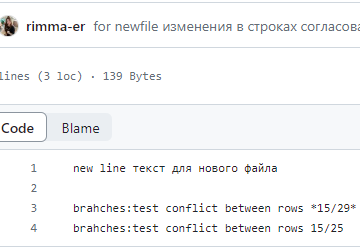


Исправляем ошибку/несоответствие данных

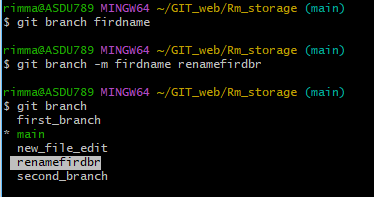
Несоответствие строк в одном файле правим вручную



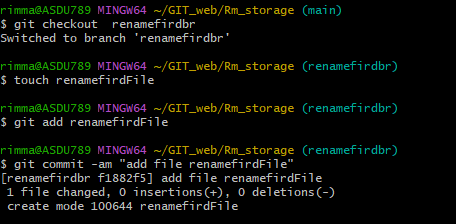


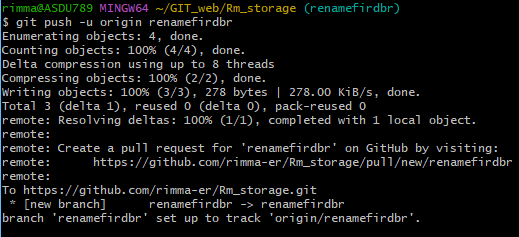


Переименовать ветку firdname в renamefirdbr (если пустая)



Переименовать ветку (если с файлами)



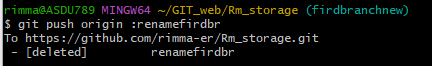


Теперь переименовываем ветку renamefirdbr в firdbranchnew

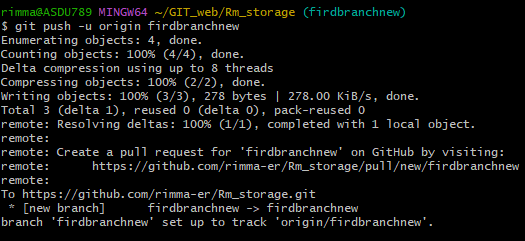


На гит хабе не переименовалась Поэтому **удаляем ветку** renamefirdbr

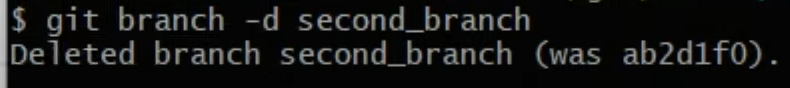
Обращаемся к ней.



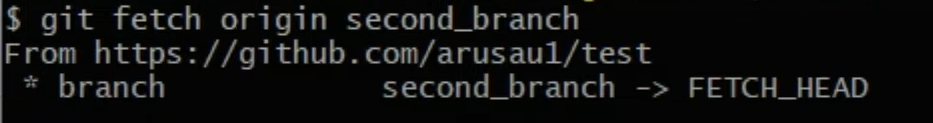
Добавляем новое название



**Удалить** ветку из **локального** репозитория.

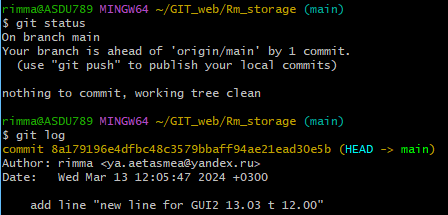


Вернуть ветку если удалена



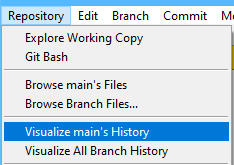
**Запуск GIT GUI**





Запуск инструмента для просмотра ветвей и связей





**GIT Flow**



Ветки

Основные:

**Master** рабочая версия продукта. Там код , который хранится на проде

**Develop** – происходит разработка.

*В начале создаем репозиторий и мастер ветку.*

*Потом делаем ее копию – Develop ветку.*

*Отдельные функции отрабатывают на Feature ветке. Потом заливают в develop. Происходит мердж develop и Feature*

*Если там все прошло хорошо появляется Release ветка на которую все добавляют. Она считается стабильной. На ней работают тестировщики. Здесь проходят только правки багов.*

*Если ошибок нет, заливаем Release в Master и Develop. Чтобы была стабильная версия приложения.*

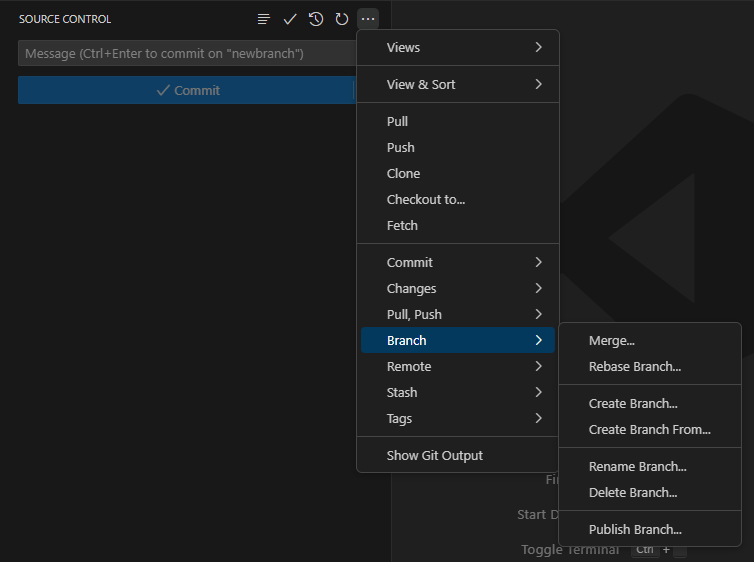
Дополнительные ветки

**Feature** дополнительные ветки для разработчиков если нужно поработать над отдельными функциями.

**HotFix** ветка для быстрого фикса багов с прода

**Release** подготовка к новому релизу

**VSCODE**



**Branch > Merge** подгружаем данные с какой-то ветки на свою ветку (к которой подключены)

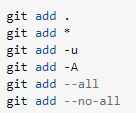
**Branch > Rebase Branch** с текущей ветки перенаправляем на другую ветку.

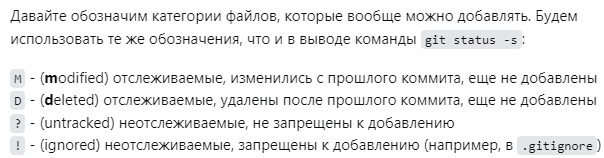
**File>Close folder** Закрыть теущий проект

Как подключиться к проекту в котором меня не было в гитхаб:

1. Создаем себе локально папку.
2. Клонируем в VScode себе репозиторий по ссылке из гитхаба
3. Указываем папку которую создали для сохранения

Тест тест тест тест





Amend last commit Изменить последний коммит

stage change Cмена этапа

Команда commit

Commit/Коммит - это способ сохранения изменений в коде. Каждый commit содержит информацию о том, что было изменено в коде и кем были внесены эти изменения. Они позволяют разработчикам отслеживать изменения в своем (или чужом) коде и возвращаться к предыдущим версиям, если это необходимо.

В автоматизации тестирования есть человекочитаемые языки, например Gherkin. Он используется для описания сценариев тест-кейсов для автотестов. Это не код приложения, а, по сути, обычный текст. Писать на таком языке может кто угодно, для этого не нужны специальные знания. Пример тестового сценария на Gherkin:

Scenario: Successful registration

Given I am on the registration page

When I enter username “admin”

And I enter password “password”

And I confirm password

And I click the register button

Then The successful registration page is opened

Я встречала проекты, где ручные тестировщики используют Gherkin для написания сценариев, а автоматизаторы уже переводят их в код.