**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

Ордена Трудового Красного Знамени федеральное государственное бюджетное учреждение высшего образования

**«Московский технический университет связи и информатики»**

**ФАКУЛЬТЕТ ИНФОРМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ**

Кафедра «МКиИТ»

**Информационные технологии и**

**программирование**

**Лабораторная работа по ИТП**

**Основы работы с системой**

**контроля версий**

Выполнил: студент группы БВТ2205

Белов Егор Олегович

**Цель работы:** изучение базовых функций и подходов к командной разработке в системе контроля версий Git.

**Задание:** определить проект (состоящий из нескольких файлов), который будет добавлен под версионный контроль, и произвести ряд операций в СКВ, включающих:

1) Создание репозитория или клонирование существующего.

2) Внесение ряда изменений в файл(ы) и выполнение коммитов (кол-во не меньше.

3) Откат изменений до одного из предыдущих коммитов двумя разными способами.

4) Создание, переименование и удаление ветки.

5) Слияние ветки с веткой мастер с решением конфликта содержания файлов.

6) Настройка игнорирования git’ом файлов определенного расширения с

использованием glob-шаблонов.

7) Вывод лога коммитов в измененном пользовательском формате.

8) Настройка удалённого репозитория: получение и отправка изменений в

репозиторий.

9) Организация совместной работы над проектом: клонирование чужого репозитория, выполнение коммитов, отправка изменений.

**1. Создание репозитория GitHub:**

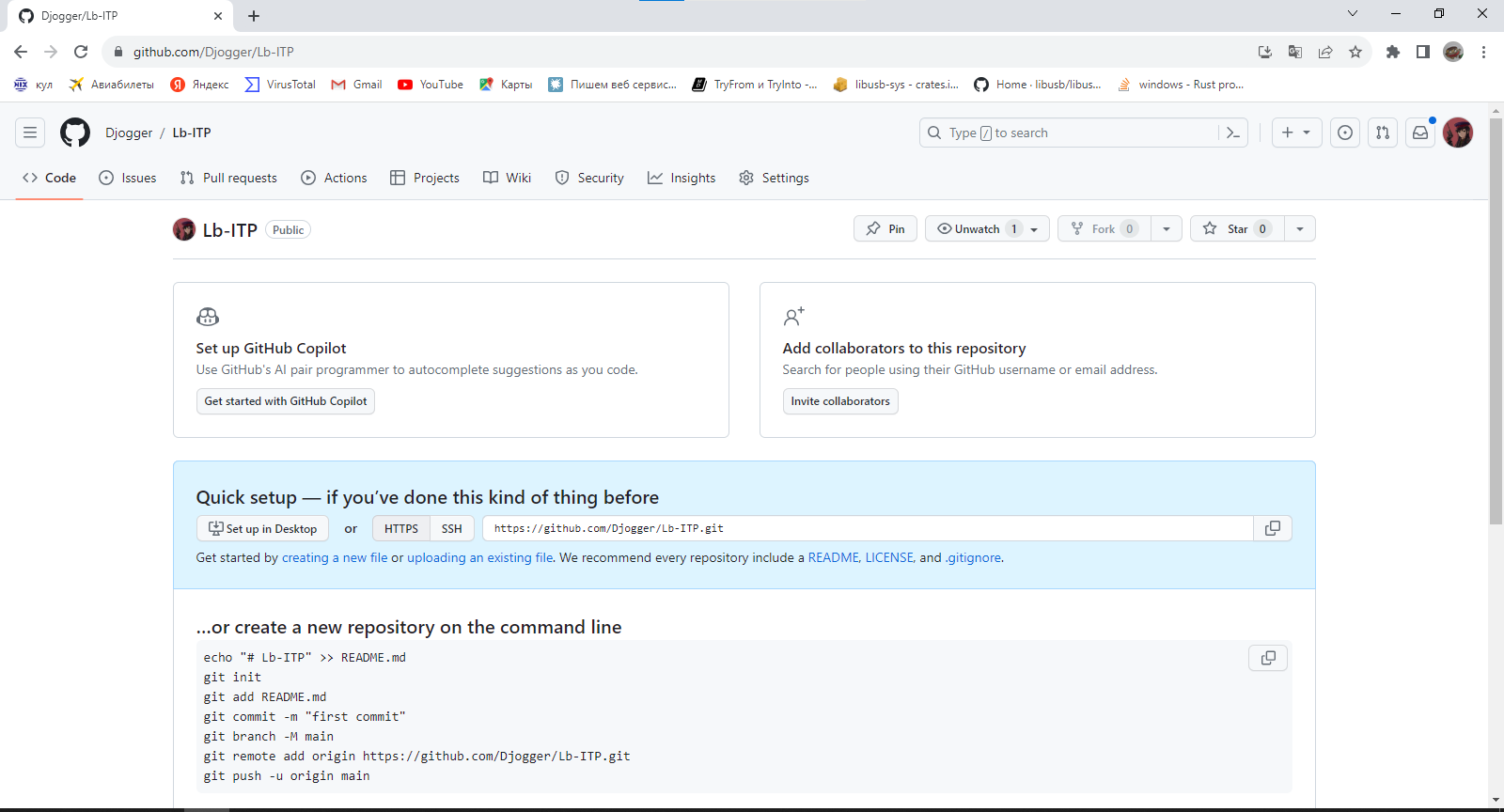
****

Рисунок 1

**2. Инициализируем репозиторий git, с помощью команды “git init”:**

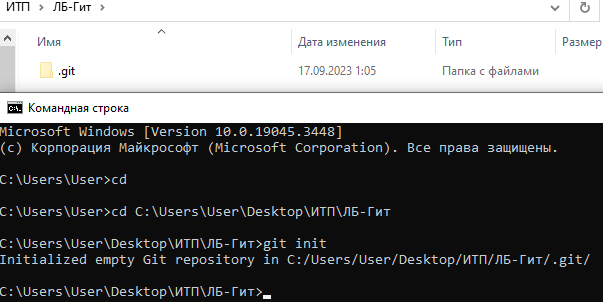
****

Рисунок 2

**3. Создал два текстовых файла для первого коммита:**

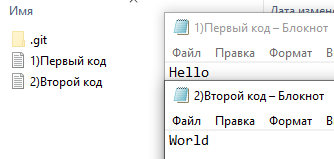


Рисунок 3

**4. Добавлю эти два файла в раздел проиндексированных файлов Git и проверю состояние рабочего каталога и раздела проиндексированных файлов:**

Команды “git add .” и “git status”:

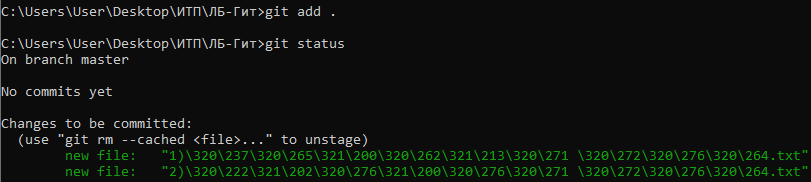


Рисунок 4

**5. Создам подключение к удалённому репозиторию:**

Команда “git remote add”:



Рисунок 5

**6. Закомментируем добавленные файлы:**

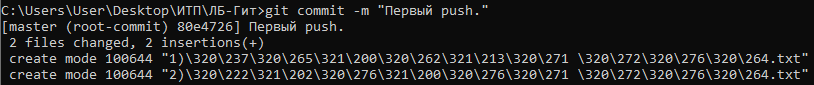


Рисунок 6

**7. Пуш кода:**

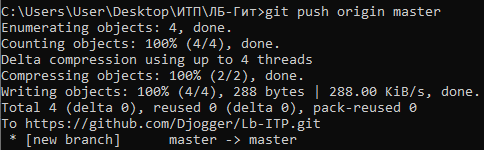


Рисунок 7

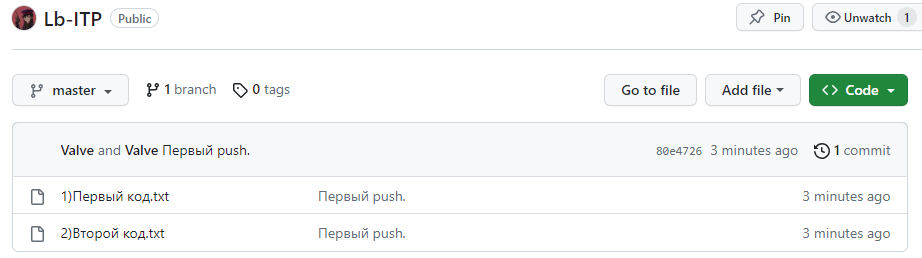


Рисунок 8

**8. Внесём изменения в файлы и запушим. Проделаем эти действия два раза:**

(Здесь рисунки прикреплять не буду, т. к. эти действия повторяю описанные выше шаги.)

**9. Получим информацию об истории коммитов:**

Команда “git log”:

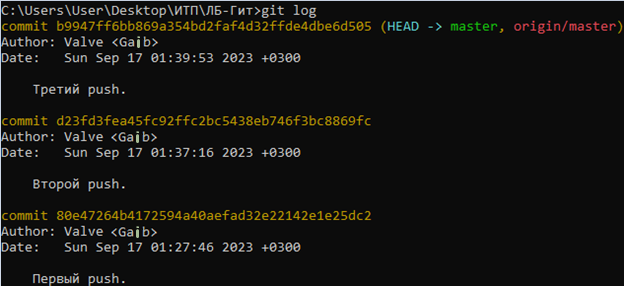


Рисунок 9

**10. Посмотрим ветки и создадим новую:**



Рисунок 10

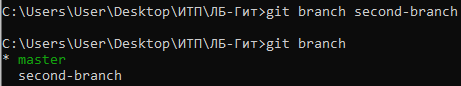


Рисунок 11

**11. Переименуем ветку “second-branch”:**

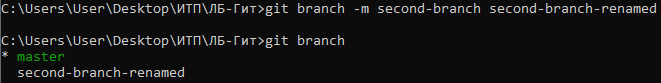


Рисунок 12

**12. Удаление ветки “second-branch-renamed”:**

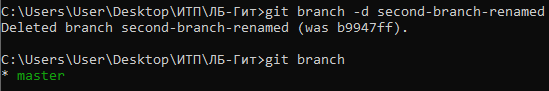


Рисунок 13

**13. Сравним два файла по хэшу коммитов:**

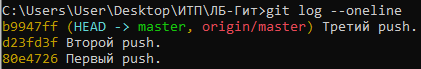


Рисунок 14

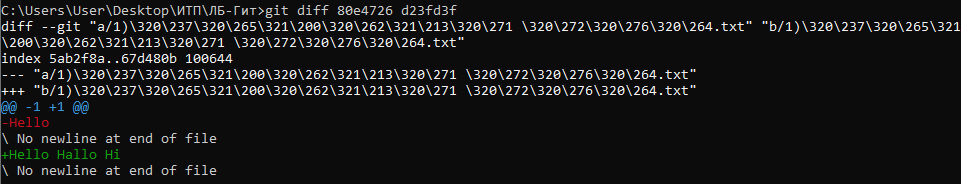


Рисунок 15

**14. Создадим снова новую ветку и перейдем на неё:**

Команда “checkout”:

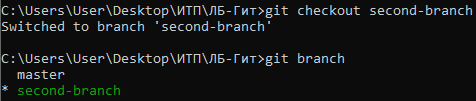


Рисунок 16

**15. Теперь мы запушим что-нибудь на вторую ветку, затем попробуем произвести слияние двух веток:**

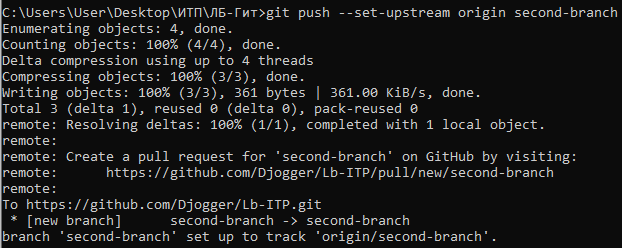


Рисунок 17

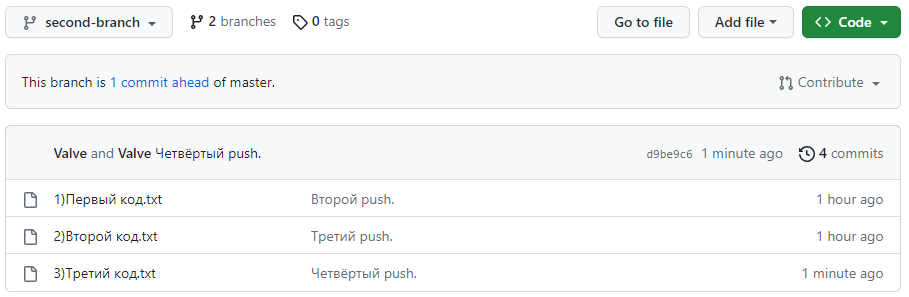


Рисунок 18

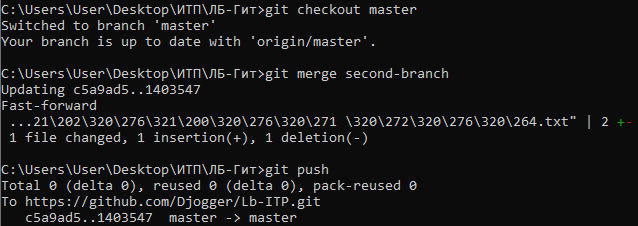


Рисунок 19

Теперь ветка ‘master’ слилась с веткой ‘second-branch’.

**16. Выполним “жёсткий сброс” или же просто откат изменений:**

**Первый способ – с помощью команды “git reset” и флагом “--hard”:**

1) Сначала создадим четвёртый файл “Code-4.txt” и поменяем файл “3)…”:

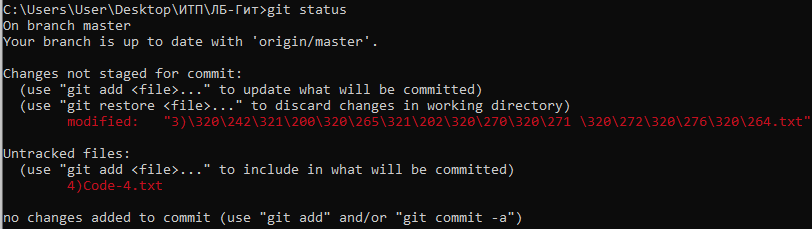


Рисунок 20

2) Добавим файл “Code-4.txt” в раздел проиндексированных файлов:

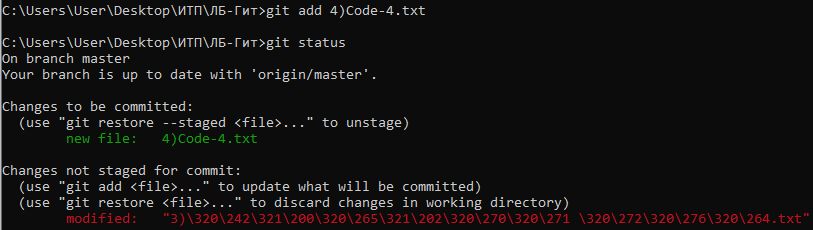


Рисунок 21

3) Проверим состояние раздела проиндексированных файлов:

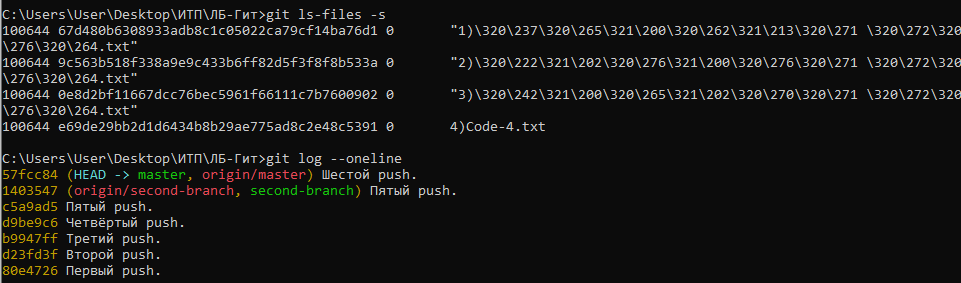


Рисунок 22

4) Осуществим сброс:

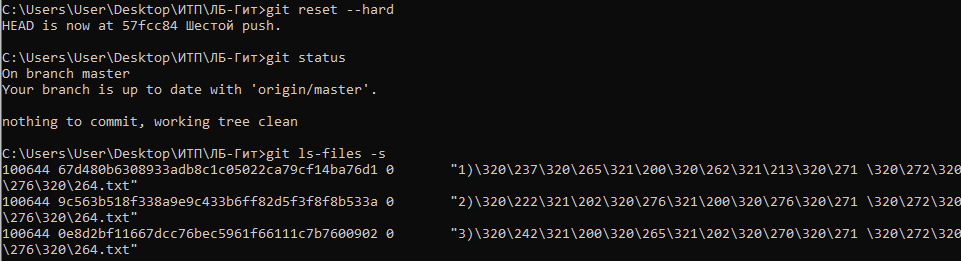


Рисунок 23

Мы теперь видим, что откат произошел на предпоследний хэш коммита, а сам файл “Code-4.txt” был удалён из локального репозитория. Также изменение в файле “3)…” было удалено и возвращено к состоянию в предыдущем коммите.

Важно понимать, что командой “git reset” стоит пользоваться только в локальной работе. В командной работе использование этой команды может создать серьезные проблемы, тем более восстановить потерянные данные будет невозможно.

**Второй способ – с помощью команды “git revert”:**

1) Проверим список коммитов, это нужно, чтобы показать отличие команды “revert” и “reset”:

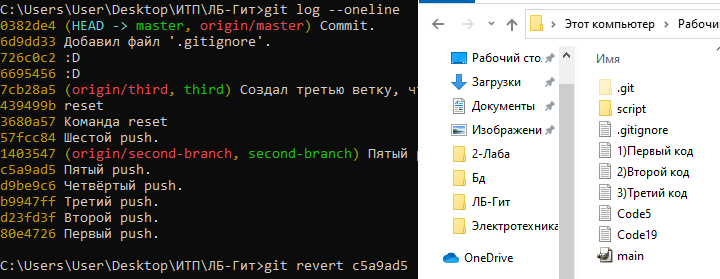


Рисунок 24

2) Выбираем хэш коммита, который мы хотим вернуть:

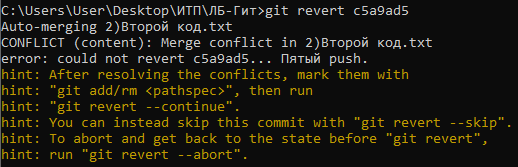


Рисунок 25

3) Тут всё так же, как и в команде “reset”:

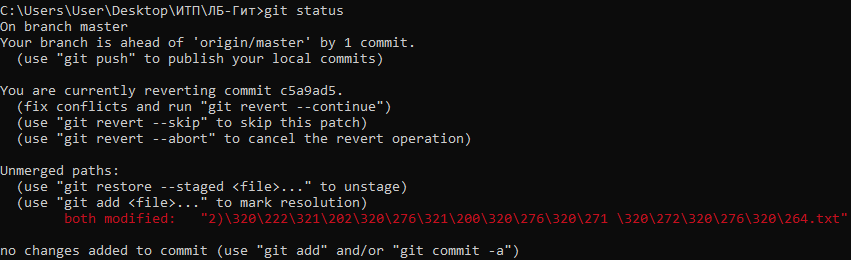


Рисунок 26

4) А вот тут уже отличие, которое является важным, т. к. из-за этого команда “revert” является необходимой для работы в команде. Когда мы откатываем изменения, то мы не удаляем коммиты после того коммита, к которому мы откатываемся, а создаём просто новый, но уже с откатом:

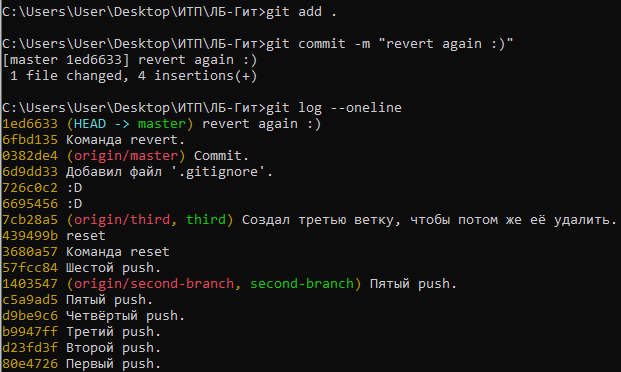


Рисунок 27

**17. Слияние ветки с веткой мастер с решением конфликта содержания файлов:**

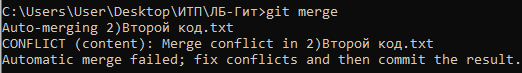
****

Рисунок 24

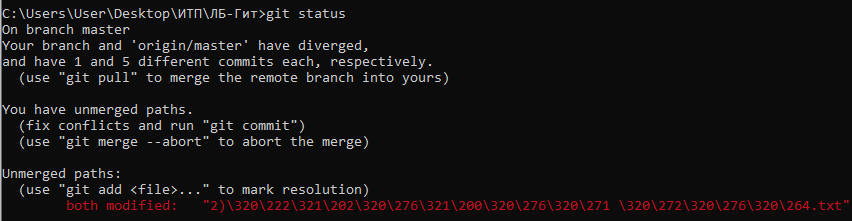
****

Рисунок 25

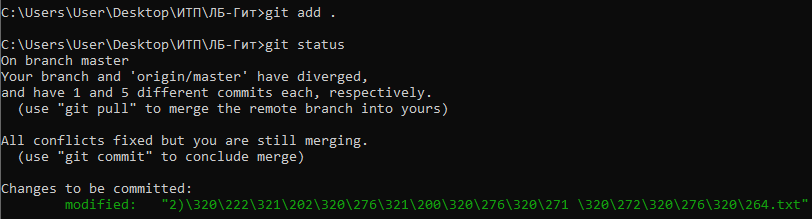
****

Рисунок 26

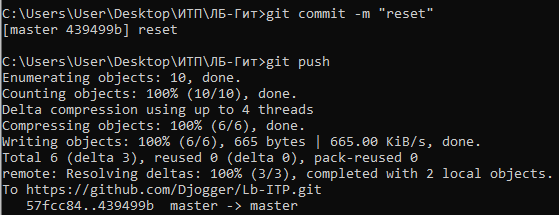
****

Рисунок 27

**18. Настройка игнорирования git-ом файлов определённого расширения с использованием glob-шаблонов:**

1. Создадим файл “.gitignore” и посмотрим структуру директории:

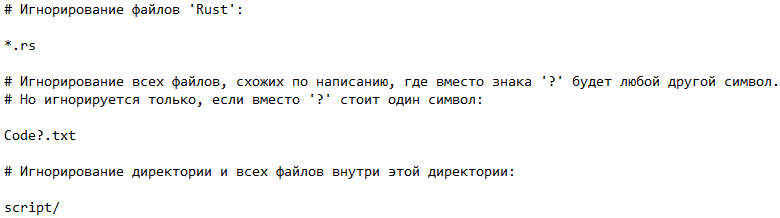


Рисунок 28

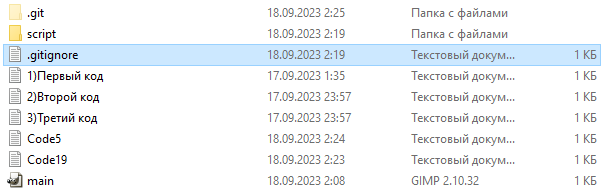


Рисунок 29

2. Будем пушить файл на GitHub. Сразу видно, что Git не видит других файлов, т. к. все файлы, показанные на рисунке 29 и при этом не запушенные на удалённый репозиторий, совпадают с файлами, которые должен игнорировать Git:

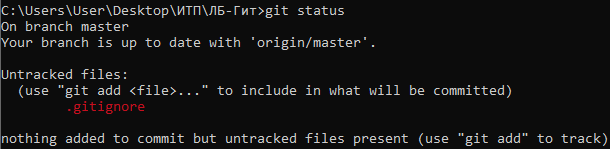


Рисунок 30

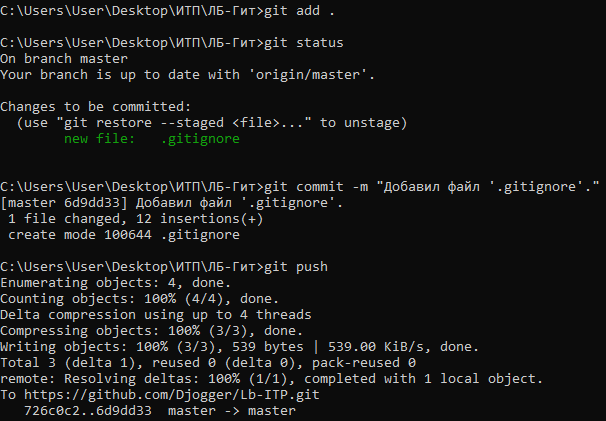


Рисунок 31

3. Затем я решил поменять имя одного файла, заменив 4 на 19, после чего Git сразу заметил его. К слову, и его я пнул на github, чтобы ему не было одиноко:

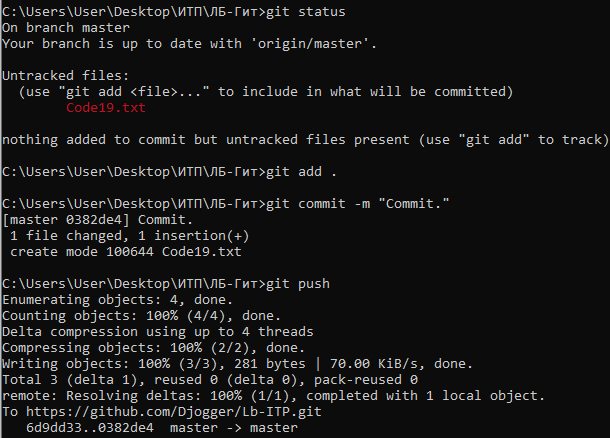


Рисунок 32

**19. Скопируем удалённый репозиторий:**

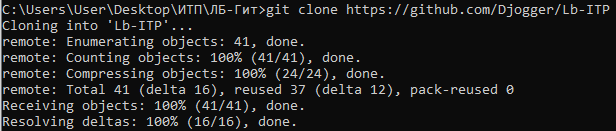


Рисунок 33

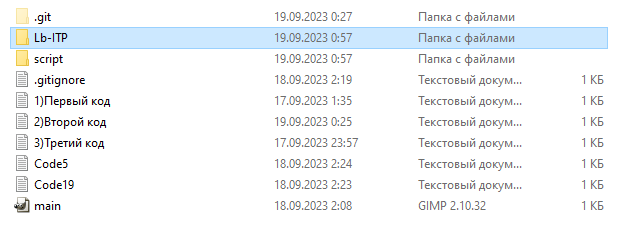


Рисунок 34

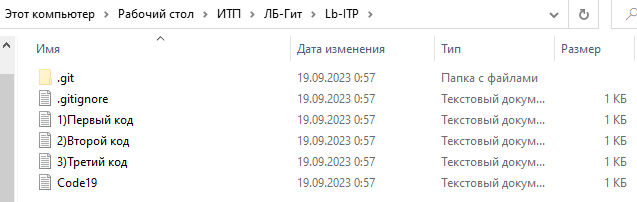


Рисунок 35

**20. Команда “git fetch”:**

1) Клонирую удалённый репозиторий:

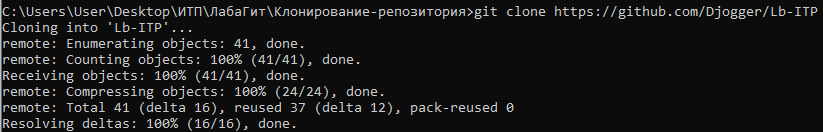


Рисунок 36

2) Захожу в скопированный репозиторий:



Рисунок 37

3) Проверяю коммиты:

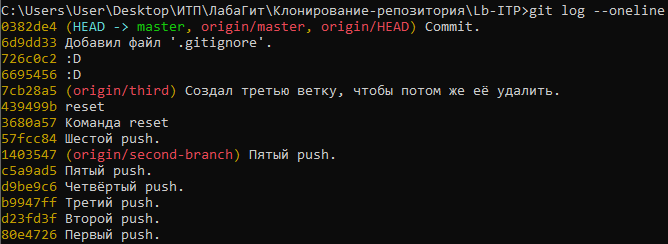


Рисунок 38

4) Перехожу к изначальной директории, где я еще не запушил два коммита:

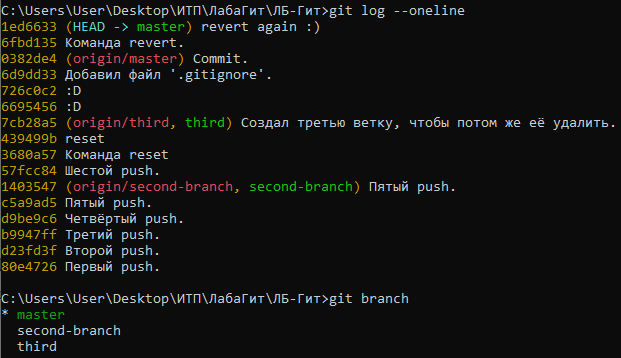


Рисунок 39

5) Пушу эти два коммита:

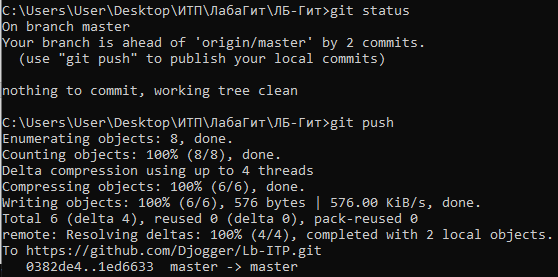


Рисунок 40

6) Возвращаюсь к клонированному репозиторию и использую команду “git fetch”, чтобы забрать изменения из удаленного репозитория:

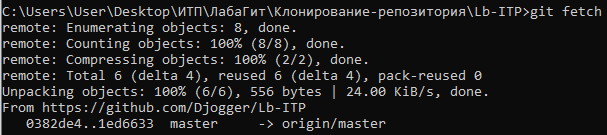


Рисунок 41

7) Использую “git merge”, чтобы объединить всё:

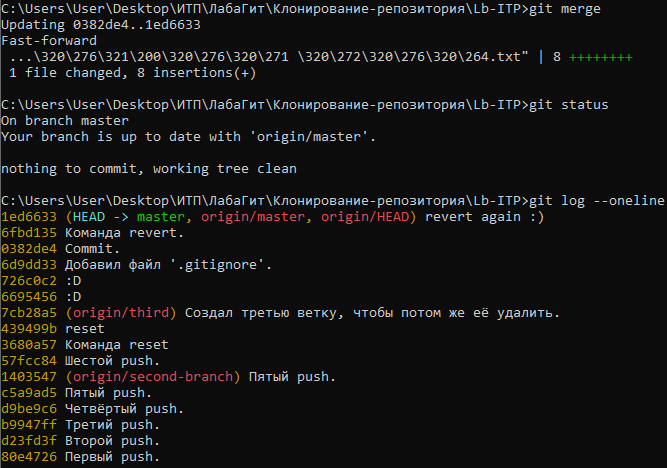


Рисунок 42

**21. Команда “git pull”:**

Команда ‘git pull’ – это то же самое, что выполнение вместе команд “git fetch” и “git merge”.

1) Создадим текстовый документ:



Рисунок 43

2) Проверим, увидел ли git новый файл:

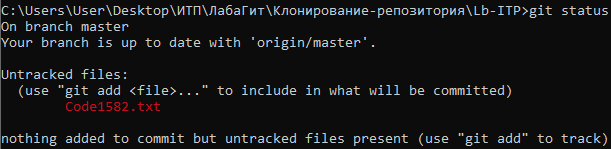


Рисунок 44

3) Запушим файл:

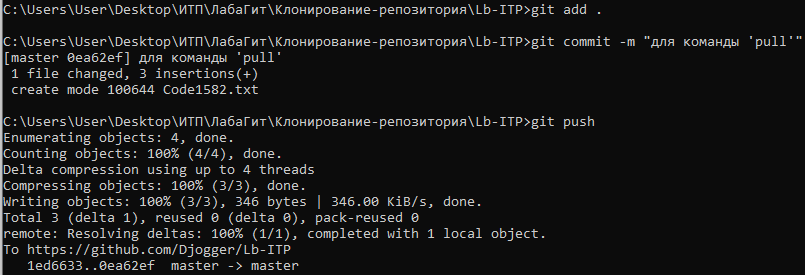


Рисунок 45

4) Теперь видим, что там, где мы пушили файл, появился коммит, но в другой директории его нет:

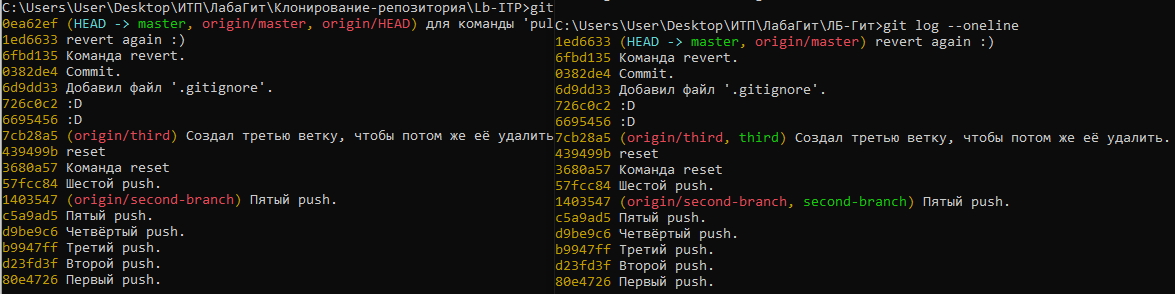


Рисунок 46

5) Заберём изменения из удалённого репозитория и объединим их с нашим локальным репозиторием:

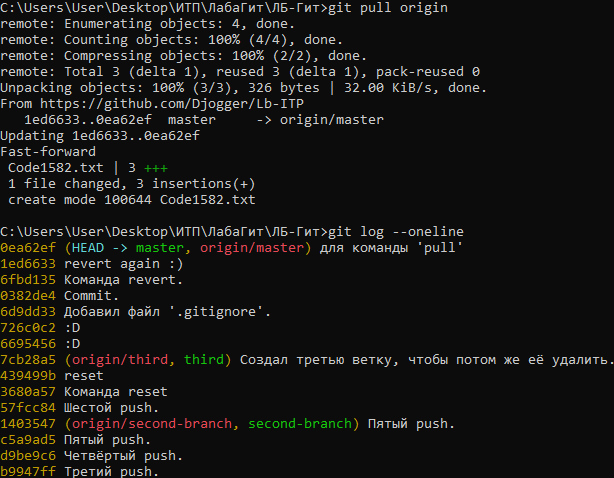


Рисунок 47

**22. Организация совместной работы над проектом: клонирование чужого репозитория, выполнение коммитов, отправка изменений.**

1) Репозиторий друга:

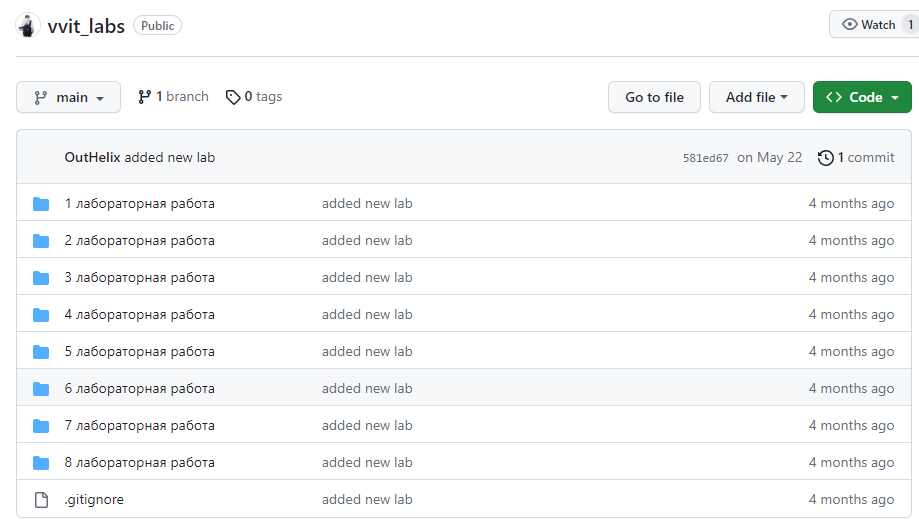


Рисунок 48

2) Клонирую репозиторий:

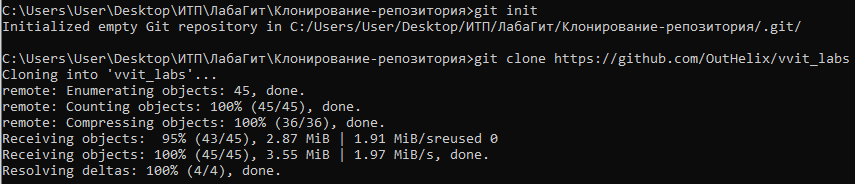


Рисунок 49

3) Добавим новый файл в репозиторий:

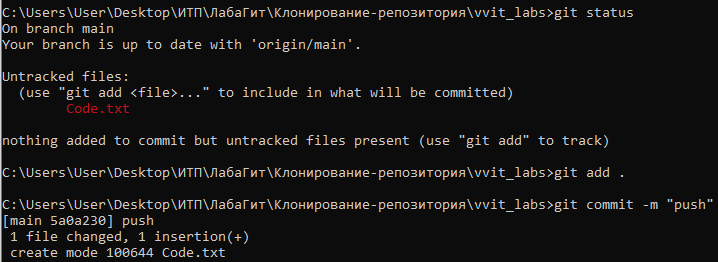


Рисунок 50

4) Принимаю приглашение:

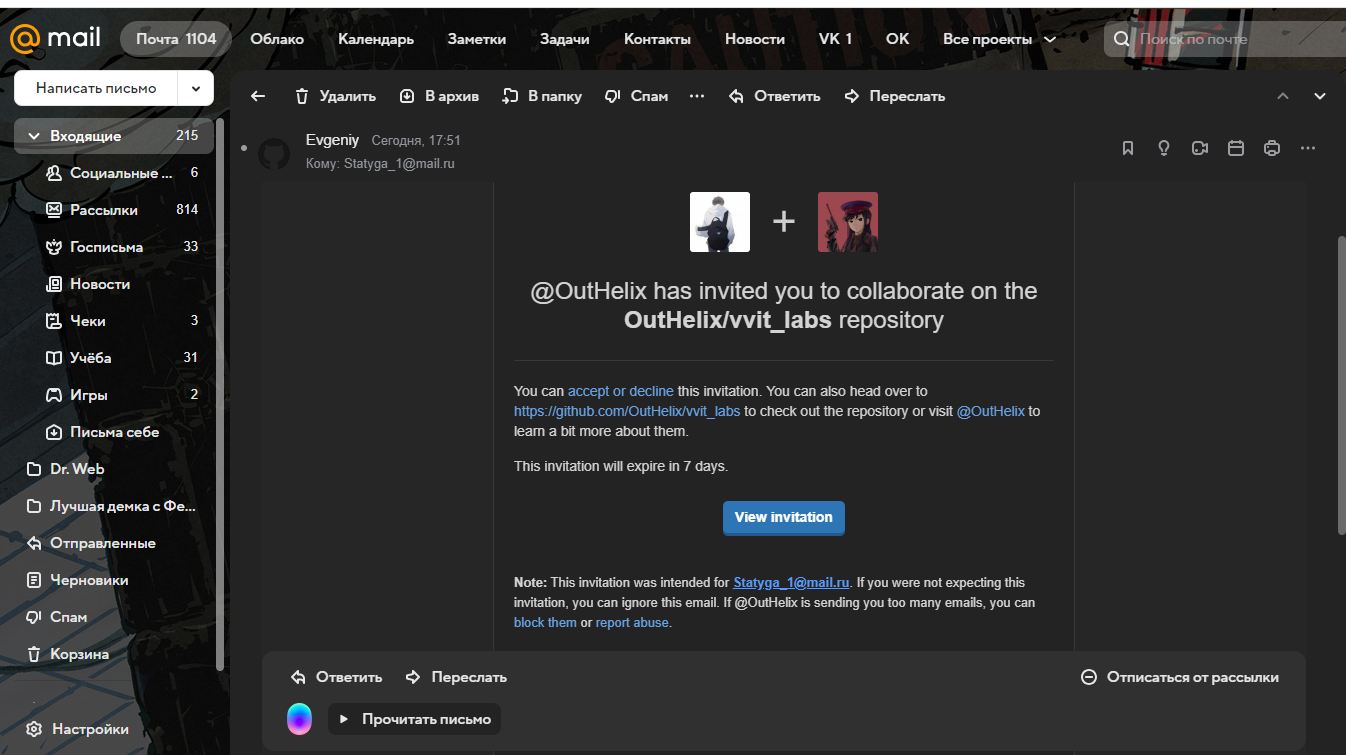


Рисунок 51

5) Пушу в репозиторий:

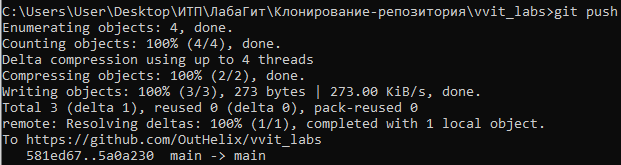


Рисунок 52

6) Проверяю репозиторий:

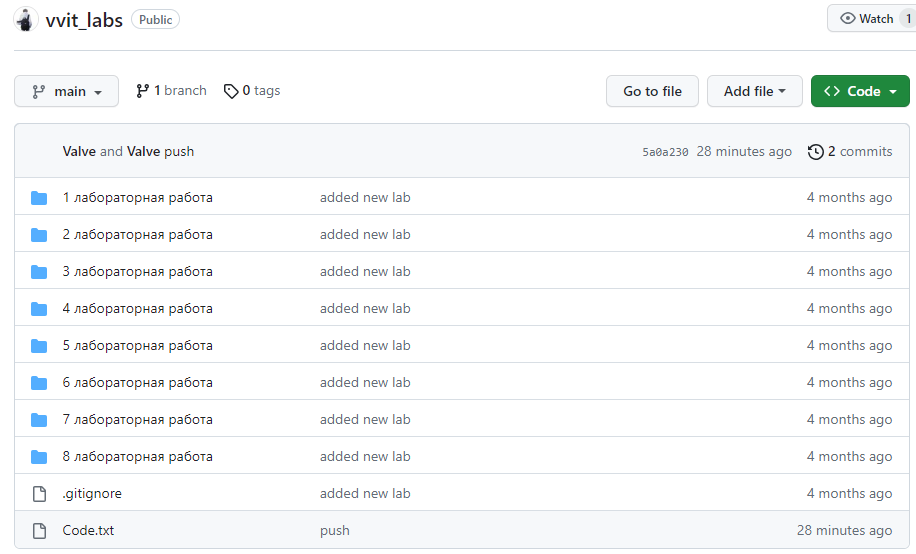


Рисунок 53

**Вывод**

В результате работы мы смогли вспомнить, как работать с git командами и репозиторием. Также мы повторили то, как получать и давать доступ к репозиторию.