МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Тульский государственный университет»

**РАБОТА С СИСТЕМОЙ КОНТРОЛЯ ВЕРСИЙ**

отчёт по практической работе №1

по курсу «Аналитика данных»

Выполнили: студенты группы 131781 Белоусова А.А.,

Майорова А.С.,

студент группы 230781 Иванов И.С.

Проверил: доцент кафедры ИБ Сафронова М.А.

Тула 2023

**Практическая работа №1  
Работа с системой контроля версий**

Цель работы: ознакомиться с возможностями системы контроля версий Git, изучить её основные команды.

Задание на работу:

1. Создать учётную запись на GitHub. Все последующие практические работы нужно будет сохранять в этой учётной записи, в других репозиториях.

2. Установить и настроить Git и клиент для него.

3. Создать новый репозиторий в своём аккаунте на GitHub и склонировать его себе на рабочую машину.

4. Выполнить следующие операции в удобном порядке: *commit*, *branch* (создать несколько дополнительных веток), *merge* (без конфликтов слияния), *merge* (с конфликтами слияния), *push*, *pull* (внести изменения в файлы через сайт GitHub и затем спуллить изменения), *pull* (с конфликтами слияния), *rebase*, *revert*, *tag* (создать несколько тегов), *stash*, *cherry-pick*, *reset*.

5. Записать итоговый результат всех изменений в центральный репозиторий на GitHub.

6. Закрыть цель и указать в отчёте ссылку на репозиторий.

Ход работы:

1. Создадим учётную запись *1MAnastasiA* на GitHub (рис.1).

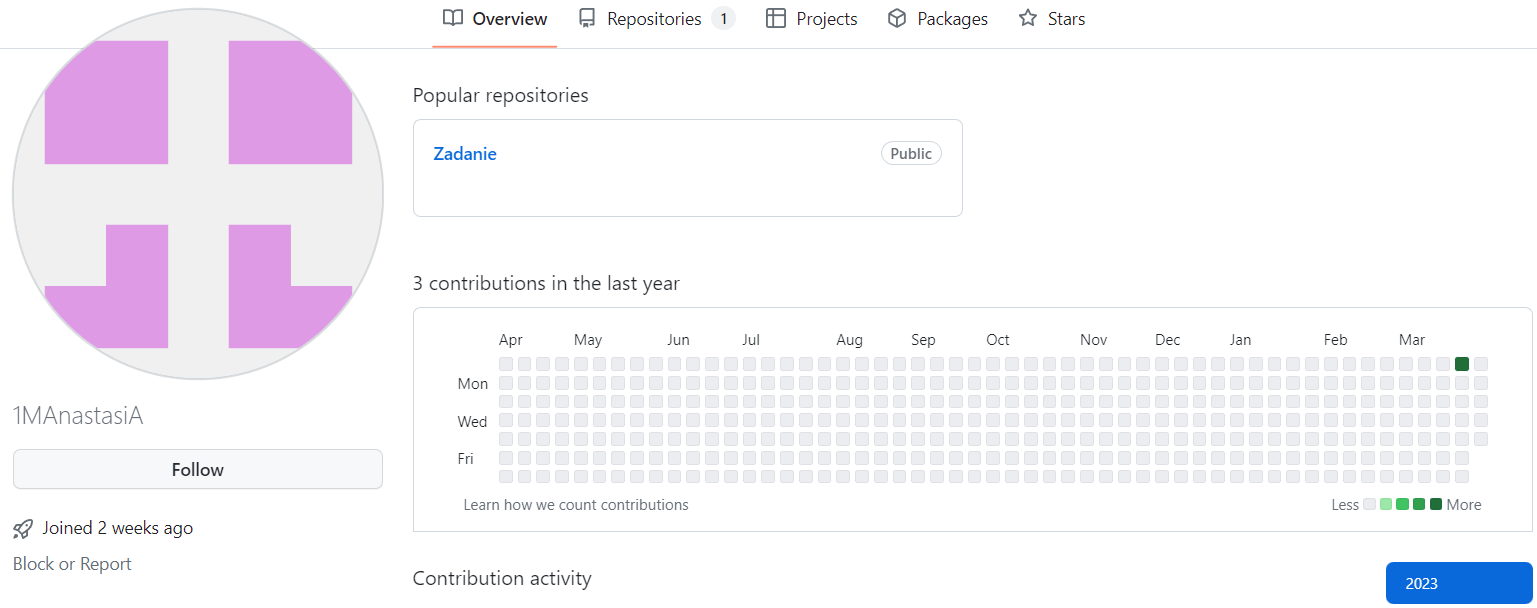


Рисунок 1 — Учётная запись *1MAnastasiA* на GitHub

2. Установим и настроим Git (рис.2) и клиент Sourcetree (рис.3) для него.

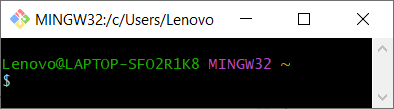


Рисунок 2 — Командная оболочка Git

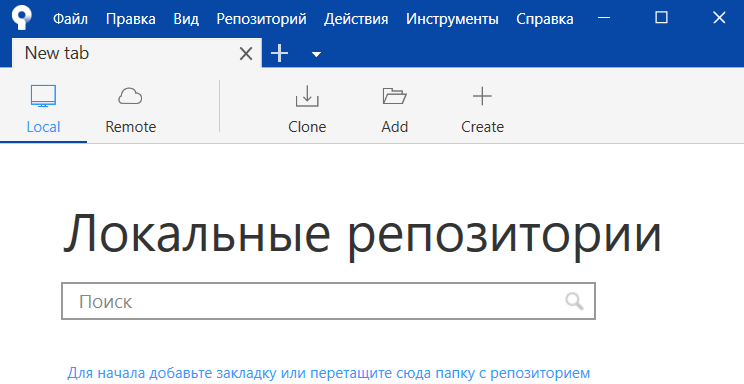


Рисунок 3 — Главное окно клиента Sourcetree

3. Создадим новый репозиторий *Zadanie* в нашем аккаунте на GitHub (рис.4) и склонируем его себе на рабочую машину (рис.5).

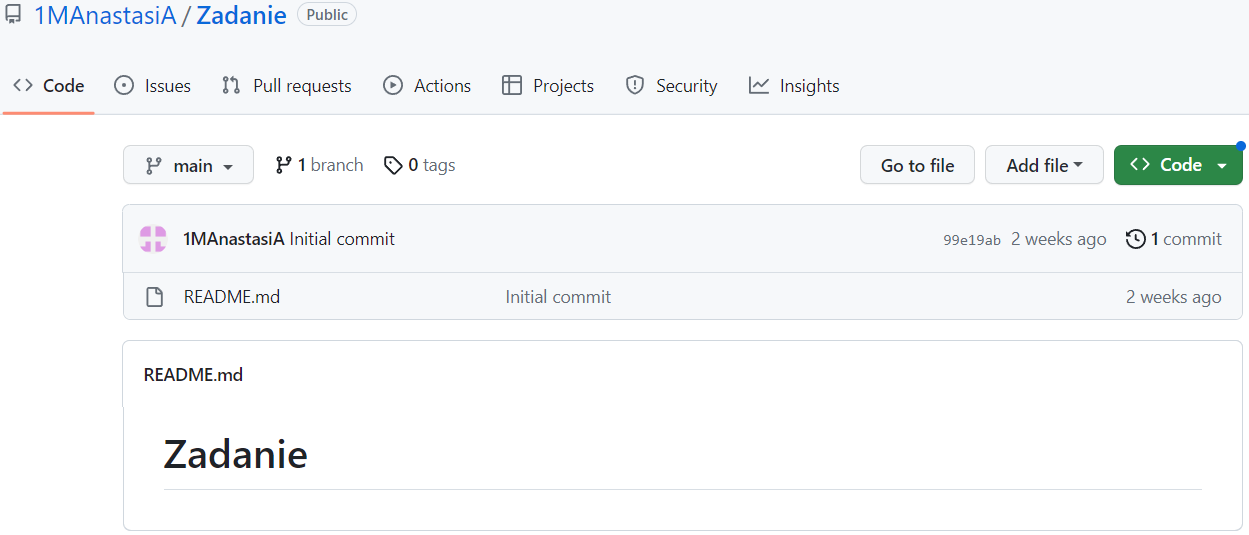


Рисунок 4 — Репозиторий *Zadanie* в учётной записи *1MAnastasiA*

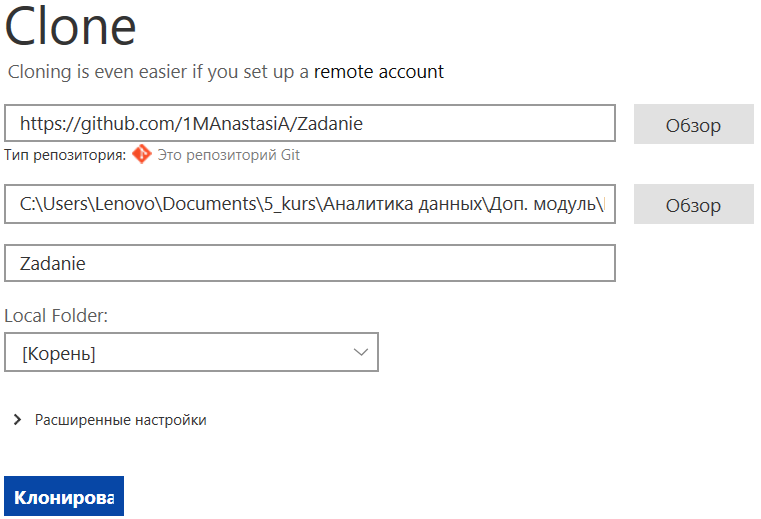


Рисунок 5 — Клонирование репозитория на рабочую машину

4. Ознакомимся с основными командами Git.

Команда *git commit* создаёт коммит. Коммит — это зафиксированный набор изменений, который показывает, какие файлы изменились и что именно в них изменилось. Например, мы хотим добавить текст монолога «Быть или не быть» из пьесы Уильяма Шекспира «Гамлет» (файл *Shakespeare.txt*). Чтобы сохранить изменения, нужно их закоммитить, но перед этим сообщить Git, какие именно файлы мы хотим положить в коммит. Команда *git add* добавляет файлы к коммиту. Затем создаём сам коммит. Флаг *-m* задаёт сообщение коммита. Оно нужно, чтобы описать изменения, которые делает данный коммит. Результаты выполнения команд представлены на рис.6.

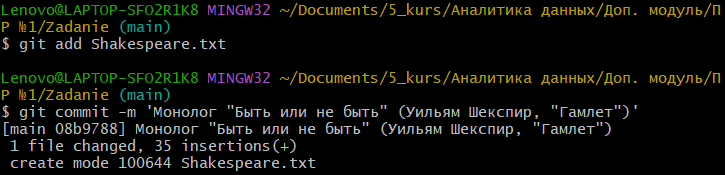


Рисунок 6 — Создание коммита

Команда *git log* показывает историю коммитов (рис.7).

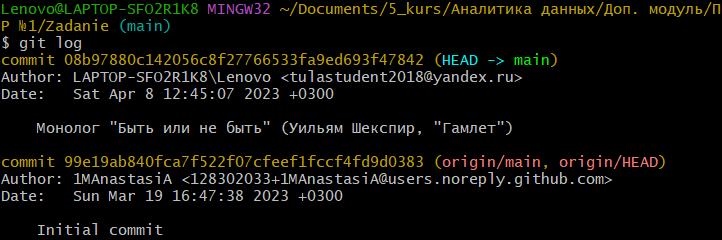


Рисунок 7 — История коммитов

Команда *git push* используется для выгрузки содержимого локального репозитория в удалённый репозиторий (рис.8 — 9). Она позволяет передать коммиты из локального репозитория в удалённый. Синтаксис команды: *git push <remote> <branch>*, где *remote* — имя удалённого репозитория, *branch* — имя ветки, которую хотим запушить.

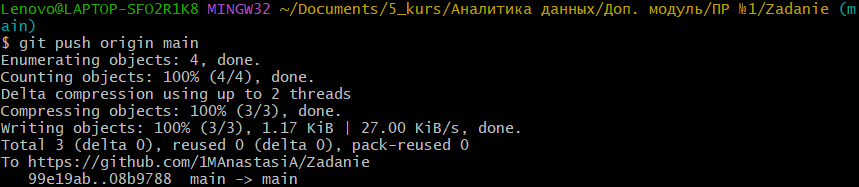


Рисунок 8 — Выгрузка содержимого локального репозитория

в удалённый репозиторий

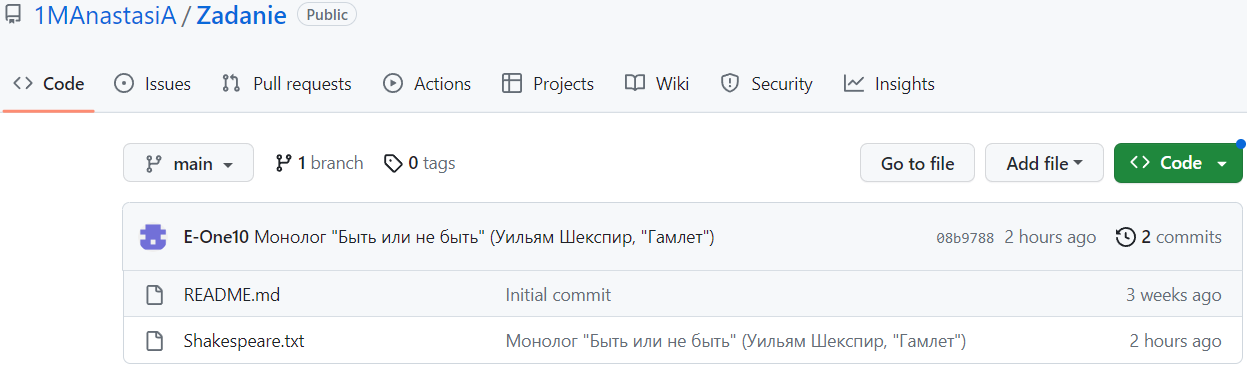


Рисунок 9 — Результат выгрузки

С помощью команды *git branch* можно добавлять новые ветки, перечислять и переименовывать существующие и удалять их. Например, создадим ветки *sub1*, *sub2* и *sub3* (рис.10). Затем просмотрим список всех (как локальных, так и удалённых) веток, используя флаг *-a* (рис.11).

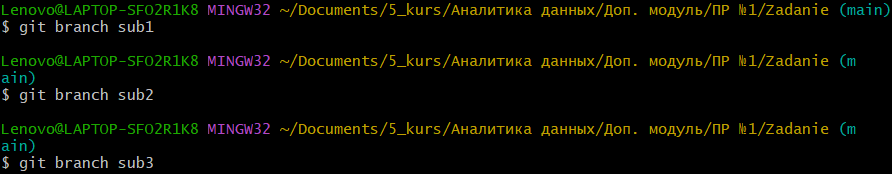


Рисунок 10 — Создание новых веток

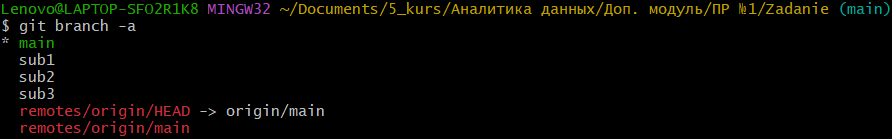


Рисунок 11 — Список всех веток

Наконец, добавим новые ветки в удалённый репозиторий, используя команду *git push -u origin <branch>*, где *branch* — имя ветки (рис.12 — 13). Флаг *-u* означает, что локальная ветка и соответствующая ей удалённая ветка будут связаны между собой.

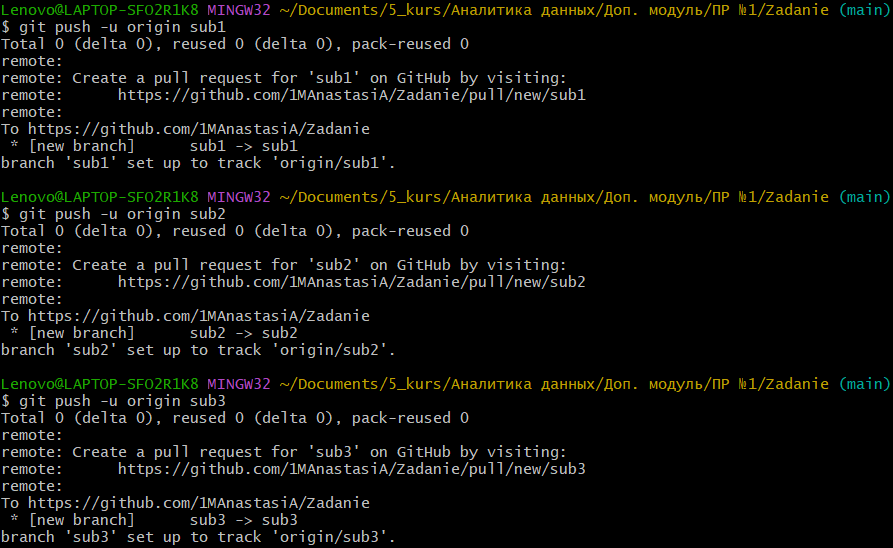


Рисунок 12 — Добавление новых веток в удалённый репозиторий, ч.1

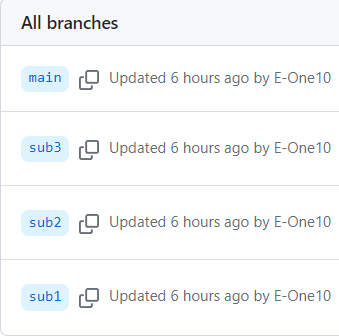


Рисунок 13 — Добавление новых веток в удалённый репозиторий, ч.2

С помощью команды *git checkout* переключимся на ветку *sub1* и создадим на ней новый коммит (рис.14 — 15).

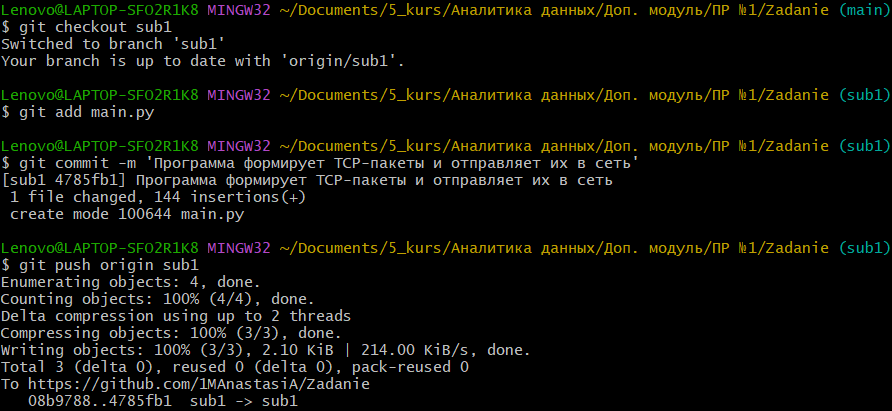


Рисунок 14 — Создание нового коммита на ветке *sub1*, ч.1

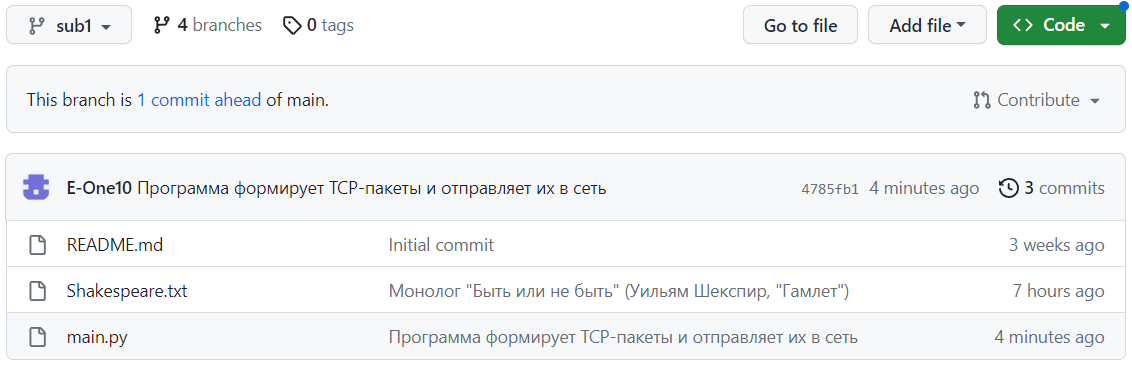


Рисунок 15 — Создание нового коммита на ветке *sub1*, ч.2

Команда *git reset* переносит указатель ветки вместе с указателем *HEAD* на переданный коммит. Данная команда позволяет удалить коммит или группу коммитов из репозитория. Существуют три опции, которые можно использовать с командой *git reset*:

* *--soft* — удаление коммитов из репозитория (без изменения рабочей директории);
* *--mixed* — удаление коммитов из репозитория и очистка индекса (это специальная промежуточная область, в которой хранятся изменения файлов на пути от рабочей директории до репозитория) (без изменения рабочей директории);
* *--hard* — удаление коммитов из репозитория, очистка индекса и внесение изменений в рабочую директорию.

Например, отменим последние два коммита и удалим все изменения в файлах на ветке *sub1* (рис.16 — 17).

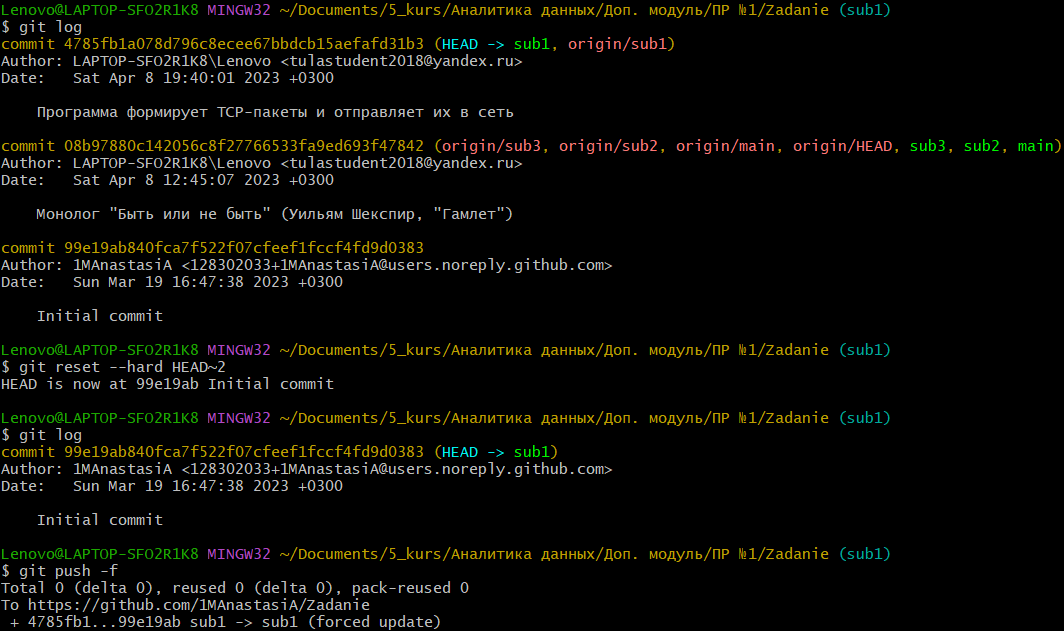


Рисунок 16 — Отмена последних двух коммитов на ветке *sub1*, ч.1

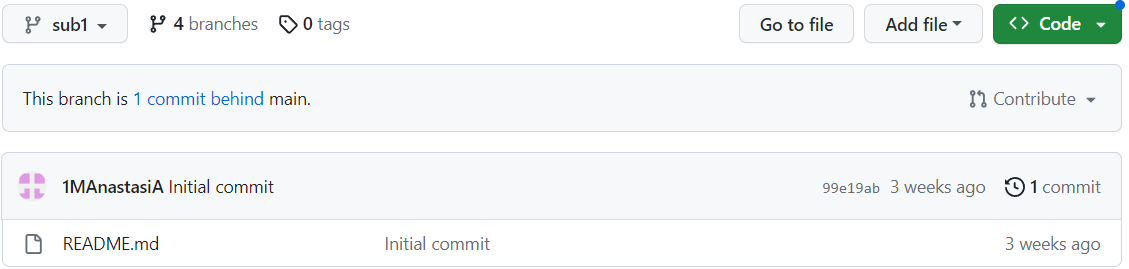


Рисунок 17 — Отмена последних двух коммитов на ветке *sub1*, ч.2

Команда *git cherry-pick* позволяет скопировать коммит из одной ветки в другую. Например, скопируем коммит с хешем 08b9788 из ветки *main* в ветку *sub1* (рис.18 — 19).

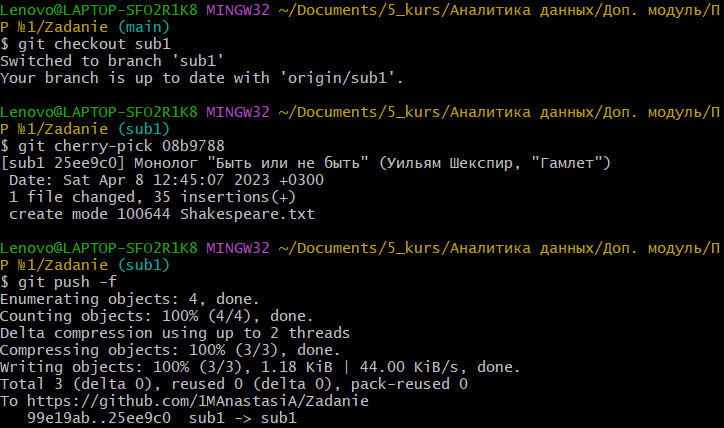


Рисунок 18 — Копирование коммита из ветки *main* в ветку *sub1*, ч.1

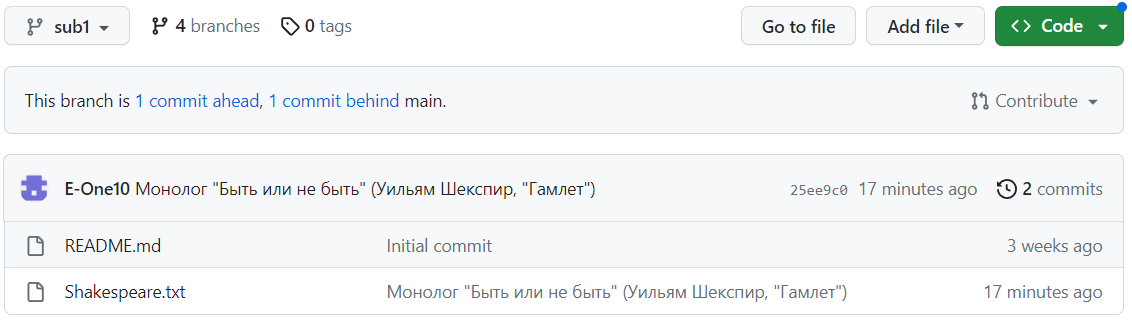


Рисунок 19 — Копирование коммита из ветки *main* в ветку *sub1*, ч.2

С помощью команды *git checkout* переключимся на ветку *sub2* и создадим на ней новый коммит (рис.20 — 21).

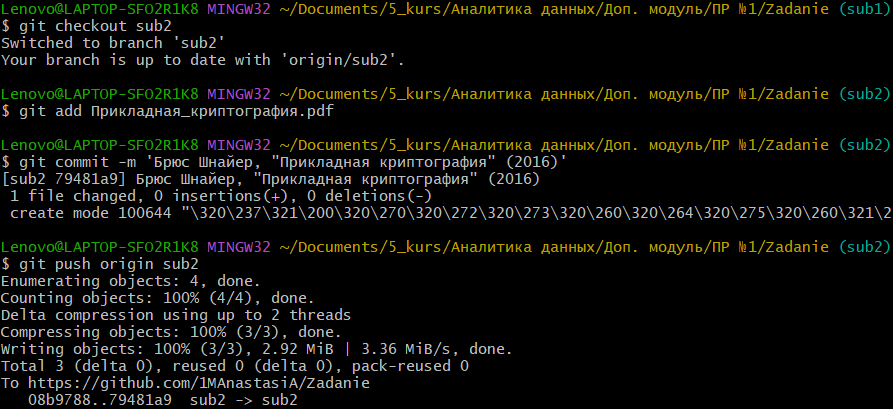


Рисунок 20 — Создание нового коммита на ветке *sub2*, ч.1

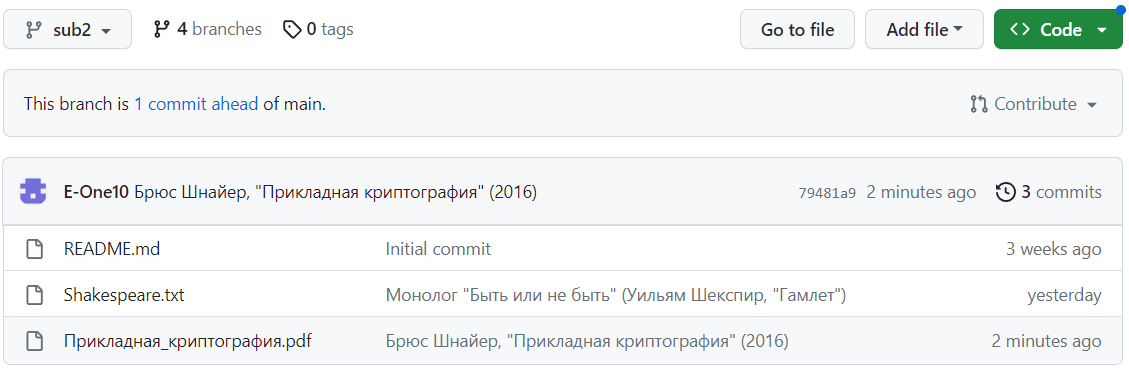


Рисунок 21 — Создание нового коммита на ветке *sub2*, ч.2

Команда *git merge* сливает изменения с переданной ветки в текущую. Например, сольём изменения с ветки *sub2* в ветку *main* (рис.22 — 23).

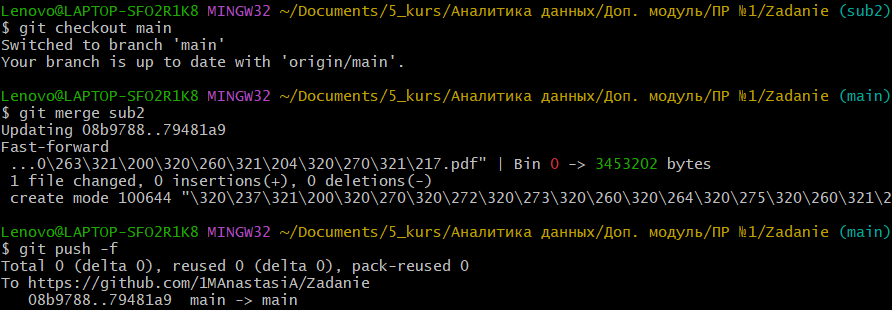


Рисунок 22 — Слияние изменений с ветки *sub2* в ветку *main*, ч.1

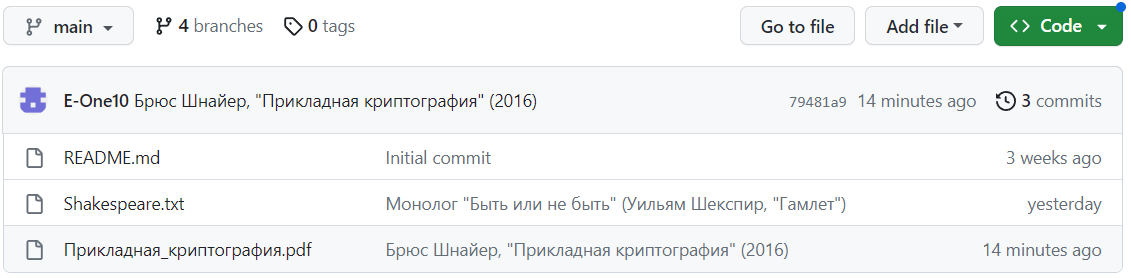


Рисунок 23 — Слияние изменений с ветки *sub2* в ветку *main*, ч.2

Команда *git tag* позволяет работать с тегами (это метки, предназначенные для обозначения важных коммитов). В Git есть два основных вида меток: аннотированные и легковесные.

Аннотированными называются такие теги, которые содержат полную информацию о версии, включая имена и e-mail разработчиков, а также дату и время создания. Чтобы создать аннотированный тег, нужно использовать флаги *-a* (идентификатор) и *-m* (комментарий) (рис.24 — 25).

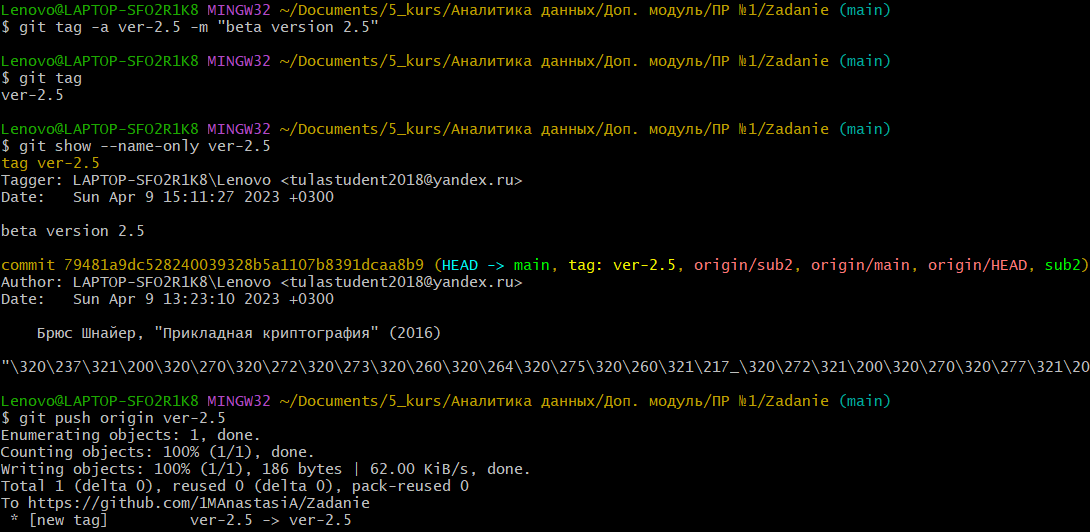


Рисунок 24 — Создание аннотированного тега, ч.1

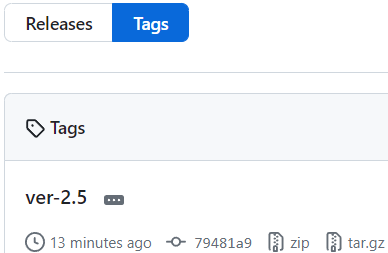


Рисунок 25 — Создание аннотированного тега, ч.2

Как видим, команда *git show* сначала выводит данные автора тега, после чего идёт комментарий, а затем данные коммита в виде контрольной  
хеш-суммы, которая состоит из 40 знаков. Также указываются автор коммита и дата его создания.

Легковесными тегами называются обычные указатели на коммиты, которые подходят для создания временных меток (фактически представляют собой сохранённые в файлы контрольные хеш-суммы коммитов). Для создания легковесных тегов флаги не используются (рис.26 — 27).

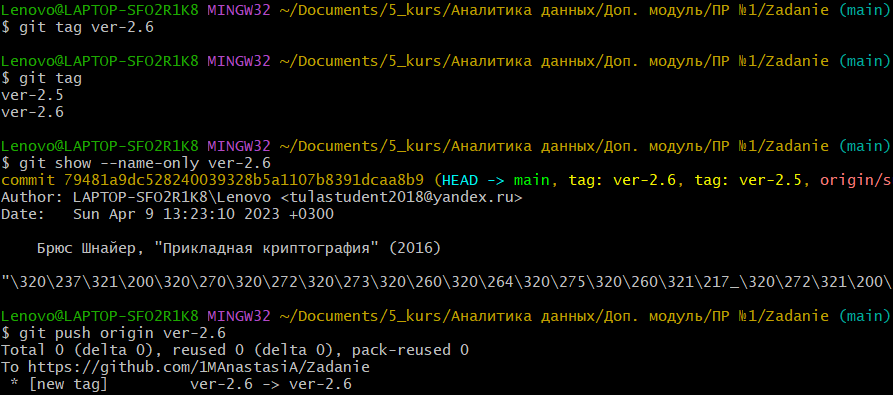


Рисунок 26 — Создание легковесного тега, ч.1

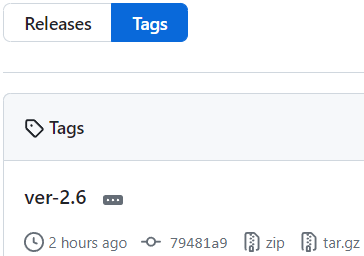


Рисунок 27 — Создание легковесного тега, ч.2

Как видим, дополнительной информации о теге нет — только данные коммита.

Команда *git revert* отменяет изменения, внесённые в переданном коммите. Фактически она сводится к созданию ещё одного коммита, который выполняет изменения, противоположные тому коммиту, который отменяется. Например, отменим изменения предпоследнего коммита, в результате чего из ветки *main* будет удалён файл *Shakespeare.txt* (рис.28 — 30).

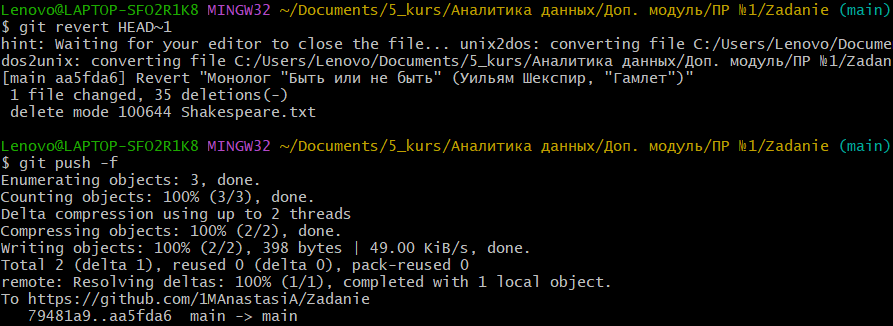


Рисунок 28 — Отмена изменений предпоследнего коммита на ветке *main*, ч.1

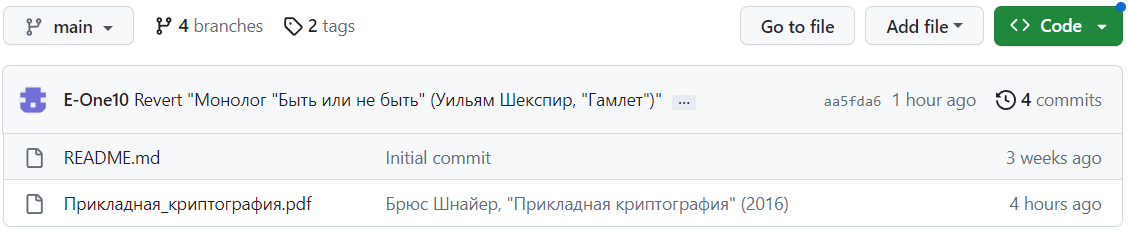


Рисунок 29 — Отмена изменений предпоследнего коммита на ветке *main*, ч.2

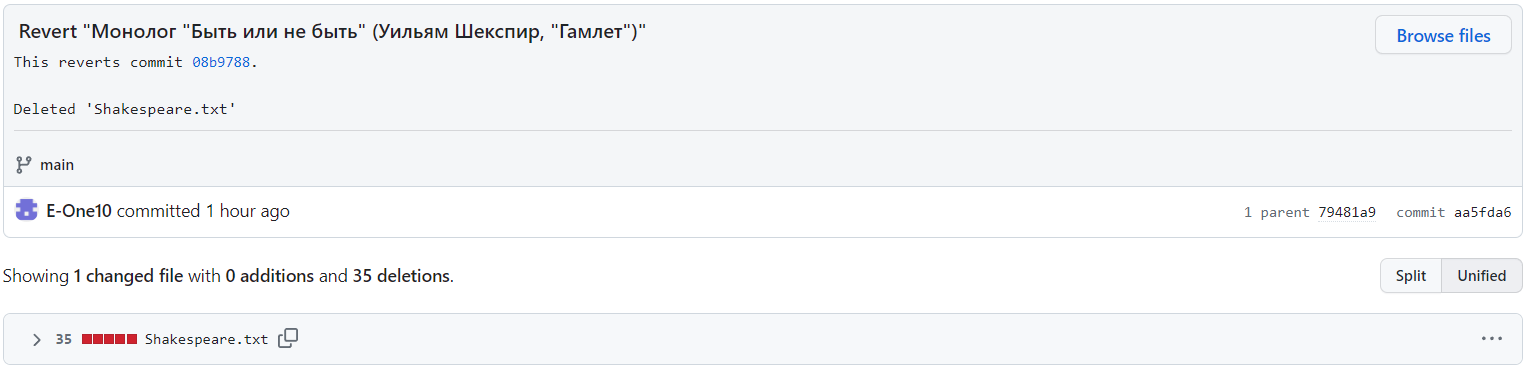


Рисунок 30 — Отмена изменений предпоследнего коммита на ветке *main*, ч.3

Команда *git rebase* перемещает все коммиты — от общего коммита двух веток до последнего коммита текущей ветки — на вершину переданной ветки. Например, сдвинем ветку *sub3* на ветку *main* (рис.31 — 33).

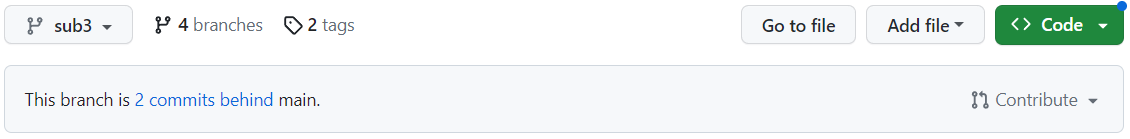


Рисунок 31 — Сдвиг ветки *sub3* на ветку *main*, ч.1

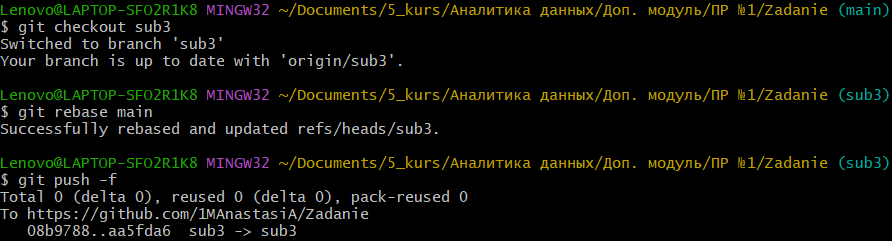


Рисунок 32 — Сдвиг ветки *sub3* на ветку *main*, ч.2

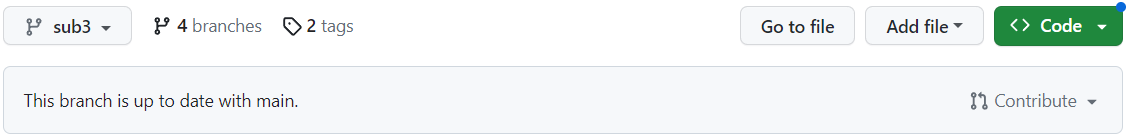


Рисунок 33 — Сдвиг ветки *sub3* на ветку *main*, ч.3

Таким образом, сдвиг привёл к линейной истории — ветке *sub3* теперь доступны новые коммиты из ветки *main*.

Команда *git stash* перемещает текущие изменения в локальную директорию, которая выполняет роль специального хранилища, т.е. скрывает эти изменения, сохраняя их отдельно, с опцией вернуть позже, когда это понадобится. Например, добавим новую строку в файл *README.md* (ветка *sub3*) и сохраним изменения локально (рис.34 — 35).

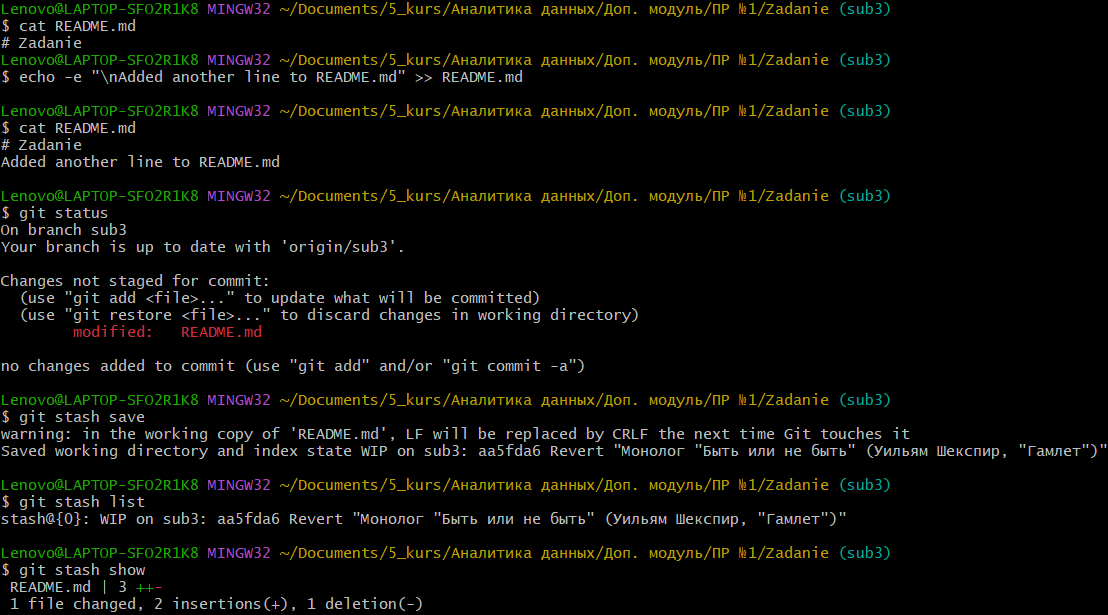


Рисунок 34 — Пример работы команды *git stash*, ч.1

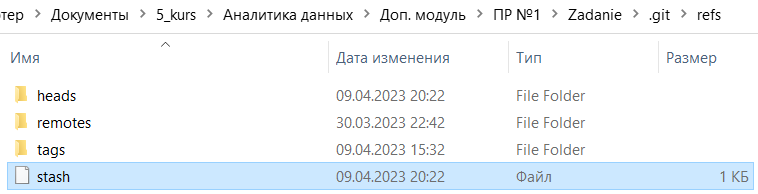


Рисунок 35 — Пример работы команды *git stash*, ч.2

Далее с помощью команды *git stash apply* достанем изменения и применим их к проекту (рис.36 — 37).

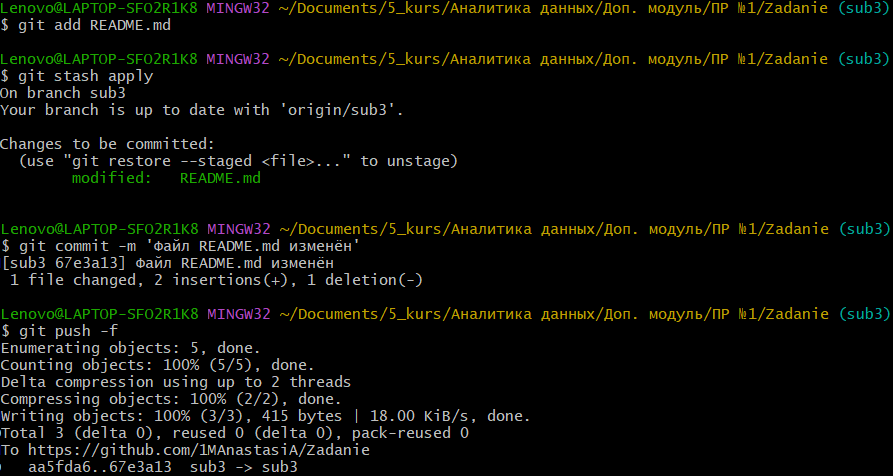


Рисунок 36 — Применение изменений к проекту, ч.1

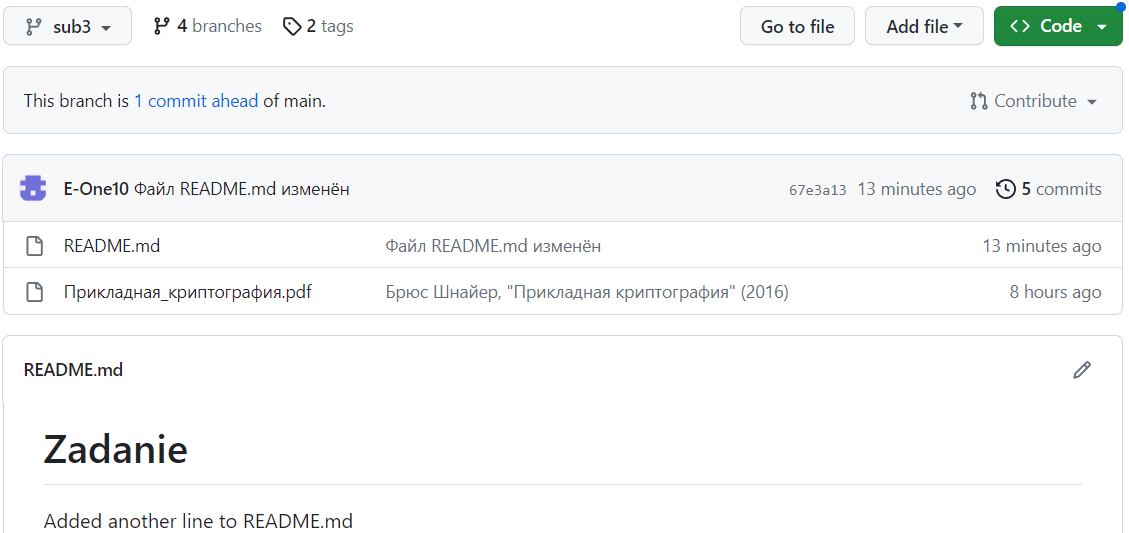


Рисунок 37 — Применение изменений к проекту, ч.2

Команда *git pull* получает изменения из переданного удалённого репозитория и обновляет рабочую копию в соответствии с удалённым репозиторием. Например, добавим в удалённый репозиторий на ветку *main* файл *Moscow.jpg* и спуллим изменения (рис.38 — 40).

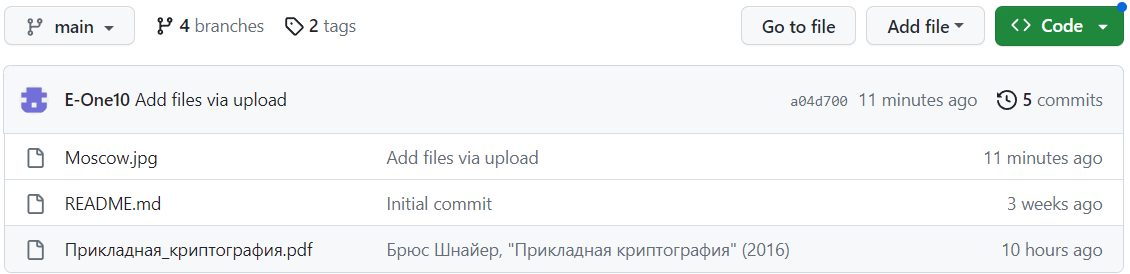


Рисунок 38 — Пример работы команды *git pull*, ч.1

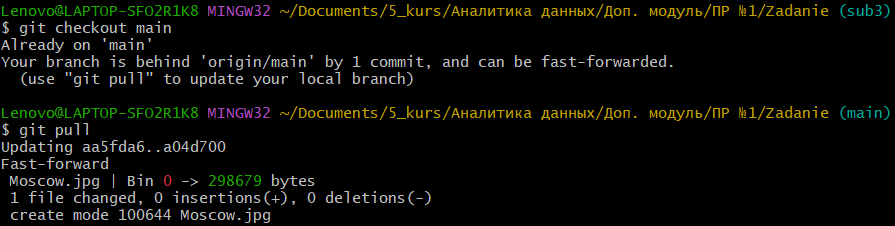


Рисунок 39 — Пример работы команды *git pull*, ч.2

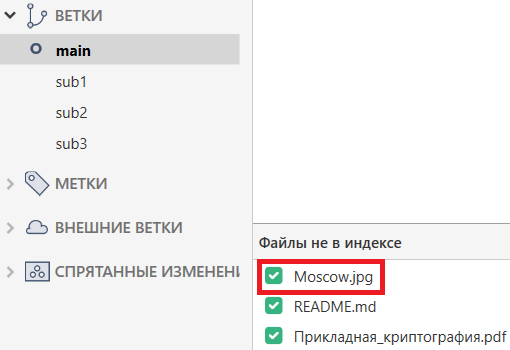


Рисунок 40 — Пример работы команды *git pull*, ч.3

Далее рассмотрим пример работы команды *git merge* в случае, когда возникает конфликт слияния. Cмоделируем конфликт для дальнейшего изучения и разрешения (рис.41 — 43).

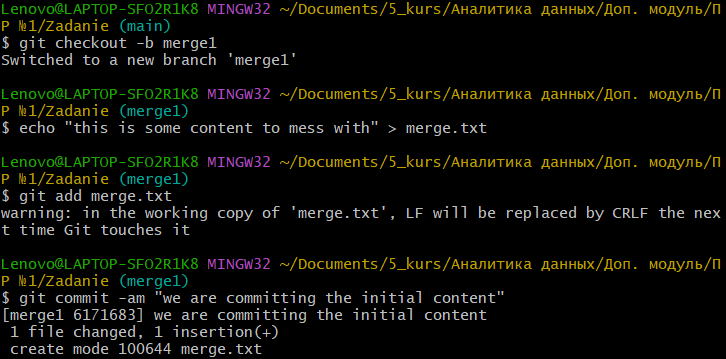


Рисунок 41 — Создание конфликта слияния (*git merge*), ч.1

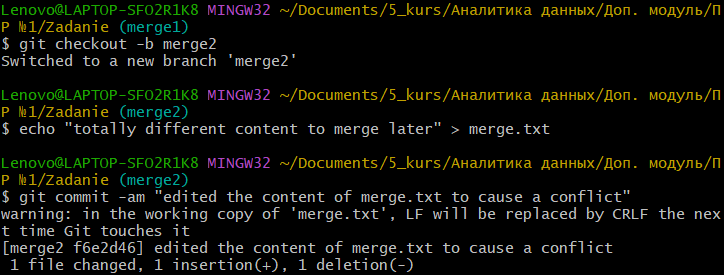


Рисунок 42 — Создание конфликта слияния (*git merge*), ч.2

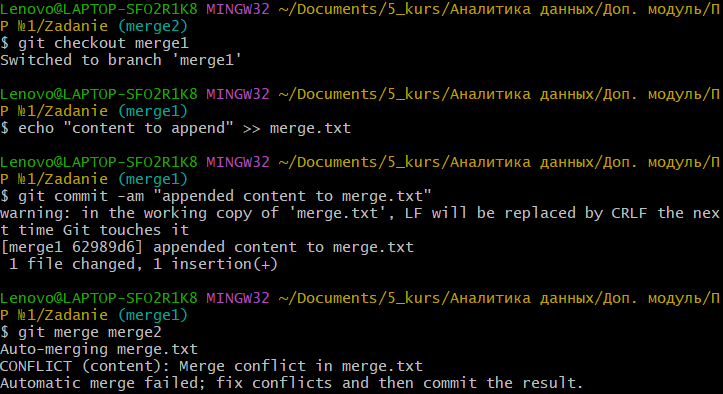


Рисунок 43 — Создание конфликта слияния (*git merge*), ч.3

На рис.41 мы создали ветку *merge1* и перешли в неё, создали новый текстовый файл *merge.txt* с некоторым содержимым, добавили его в репозиторий и выполнили коммит. На рис.42 мы создали ветку *merge2*, которая будет использоваться как конфликтующая при слиянии, и перешли в неё, перезаписали содержимое файла *merge.txt* и выполнили коммит нового содержимого. На рис.43 мы переключились на ветку *merge1*, добавили содержимое в файл *merge.txt* и сделали коммит. В итоге имеем два новых коммита: первый — в ветке *merge1*, второй — в ветке *merge2*. При попытке слить изменения с ветки *merge2* в ветку *merge1* возникает конфликт.

Чтобы получить более глубокое понимание конфликта, запустим команду *git status* (рис.44).

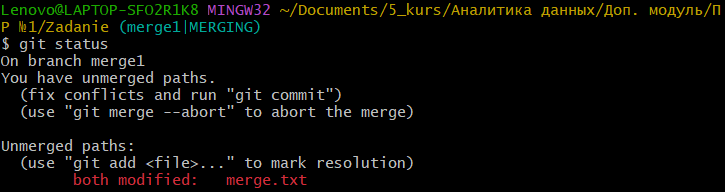


Рисунок 44 — Причина конфликта (*git merge*)

Как видим, из-за конфликта не удалось слить пути. Теперь файл *merge.txt* отображается как изменённый. Изучим этот файл и посмотрим, что изменилось (рис.45).

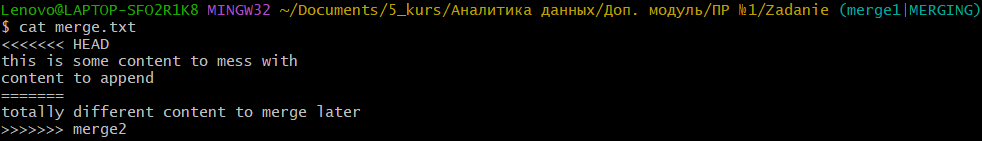


Рисунок 45 — Просмотр файла *merge.txt*

Как видим, в файле появились новые странные дополнения, которые можно рассматривать как «разделители конфликта». Строка *«=======»* является центром конфликта. Всё содержимое между этим центром и строкой *«<<<<<<< HEAD»* находится в текущей ветке *merge1*, на которую ссылается указатель *HEAD*. А всё содержимое между центром и строкой *«>>>>>>> merge2»* является содержимым ветки для слияния.

Самый простой способ разрешить конфликт — отредактировать конфликтующий файл (рис.46).

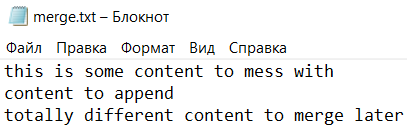


Рисунок 46 — Редактирование конфликтующего файла

После редактирования выполним команду *git add merge.txt*, чтобы добавить новое объединённое содержимое в раздел проиндексированных файлов. Для завершения слияния создадим новый коммит (рис.47).

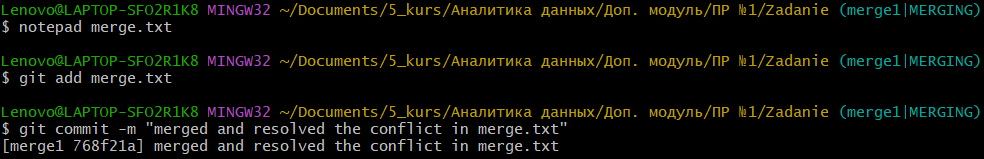


Рисунок 47 — Разрешение конфликта слияния (*git merge*)

На рис.48 — 50 приведён пример работы команды *git pull* в случае, когда возникает конфликт слияния.

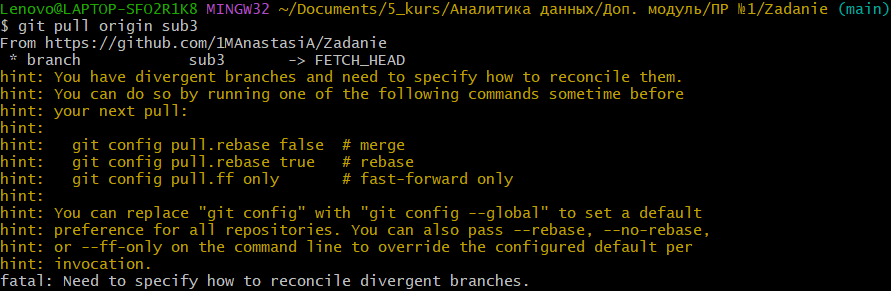


Рисунок 48 — Создание конфликта слияния (*git pull*)

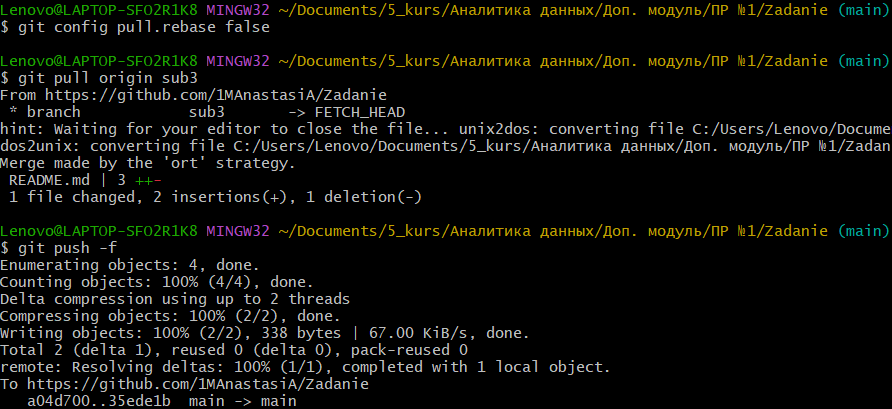


Рисунок 49 — Разрешение конфликта слияния (*git pull*)

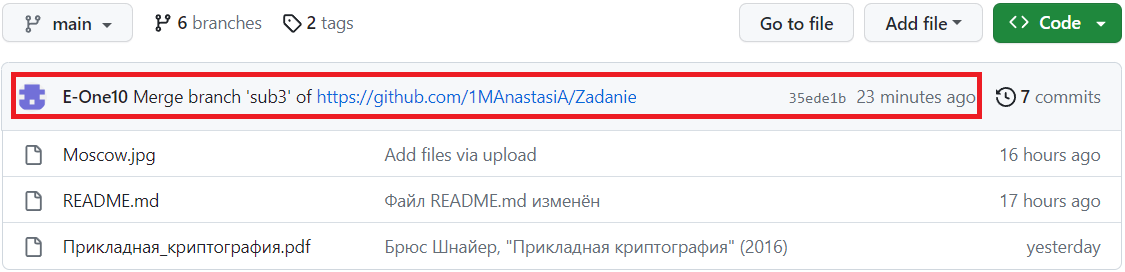


Рисунок 50 — Изменения в удалённом репозитории

5. Запишем итоговый результат всех изменений в центральный репозиторий на GitHub.

6. Ссылка на репозиторий: <https://github.com/1MAnastasiA/Zadanie>.

Вывод: в ходе практической работы были изучены возможности и основные команды системы контроля версий Git.