Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 3
дисциплины «Программирование на Python»

Выполнила:
Кубанова Ксения Олеговна
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника», очная
форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р. А.

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Тема: Основы ветвления Git

Цель: исследование базовых возможностей по работе с локальными и удаленными ветками Git.

Теоретический материал

HEAD

HEAD – это указатель, задача которого ссылаться на определенный коммит в репозитории. это указатель на коммит в вашем репозитории, который станет родителем

следующего коммита. HEAD указывает на коммит, относительного которого будет создана рабочая копия

во-время операции checkout . Переключаясь с ветки на ветку в репозитории указатель HEAD будет переключаться

между последними коммитами выбираемых ветвей.

Ветки и работа с ними

При создании новой ветки появляется новый указатель для дальнейшего перемещения.

> git branch <branch> - создаёт новую ветку.

> git log --oneline - -decorate – показывает направление веток на какие-либо коммиты.

> git checkout <branch> - переключатель веток.

> git checkout -b <branch> - создаёт и сразу переключается на ветку.

Слияние веток включает сделанные изменения одной ветки в выбранную.

$$\begin{cases} > \text{ git checkout} < first\ branch > \\ > \text{ git merge} < \text{second branch} > \end{cases}$$

Выше написанные команды работают вместе при слиянии веток. Первая – переходит на ветку, на которую нужно слить следующую, что делает уже вторая команда.

Однако иногда случаются *конфликты*. Например, в таких ситуациях, когда на разных ветках были сделаны разные изменения. Для устранения

конфликтов существует два способа: **ручной** и путём использования команды **git margetool >> <инструмент>**.

Помимо создания и управления ветками в git bash, можно так же создать ветку на удалённом репозитории, а после получать с неё изменения путём отслеживания веток. Отслеживание веток можно реализовывать со всеми ветками, которые находятся на удалённом репозитории.

> Git checkout - - track <branch> - отслеживает ветку в терминале с удалённого репозитория.

Порядок выполнения работы

Были созданы три файла, а после, соответственно, проиндексированы и закоммитены.

```
Student@PC-02-9_521 MINGW64 ~/Desktop/K/Программирование на pyth/1.3/lab1.3_pyth
/doc (main)
$ git add 1.txt

Student@PC-02-9_521 MINGW64 ~/Desktop/K/Программирование на pyth/1.3/lab1.3_pyth
/doc (main)
$ git commit -m "add 1.txt file"
[main 93265d6] add 1.txt file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 doc/1.txt
```

Рисунок 1 – индексация файла 1

```
Student@PC-02-9_521 MINGW64 ~/Desktop/K/Программирование на pyth/1.3/lab1.3_pyth
/doc (main)
$ git add 2.txt

Student@PC-02-9_521 MINGW64 ~/Desktop/K/Программирование на pyth/1.3/lab1.3_pyth
/doc (main)
$ git add 3.txt

Student@PC-02-9_521 MINGW64 ~/Desktop/K/Программирование на pyth/1.3/lab1.3_pyth
/doc (main)
$ git commit -m "add 2.txt and 3.txt"
[main 8d93be0] add 2.txt and 3.txt
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 doc/2.txt
 create mode 100644 doc/3.txt
```

Рисунок 2 – индексация файлов 2 и 3

После была создана новая ветка.

```
Student@PC-02-9_521 MINGW64 ~/Desktop/K/Прогр
/doc (main)
$ git branch my_first_branch
```

Рисунок 3 – создание ветки my_first_branch

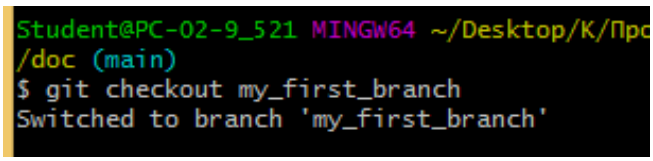

Рисунок 4 – переход на новую ветку

Был создан новый файл in_branch.txt.

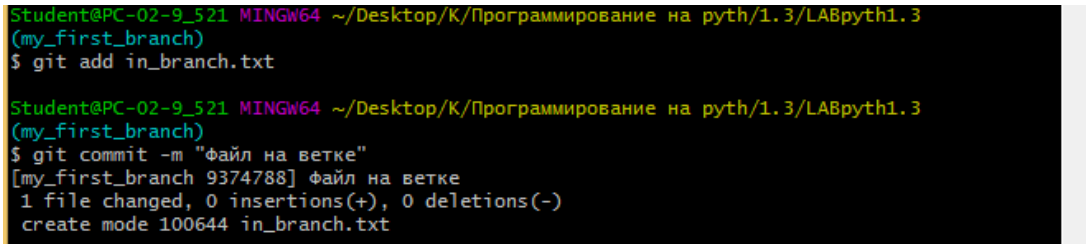

Рисунок 5 – индексация файла in_branch.txt

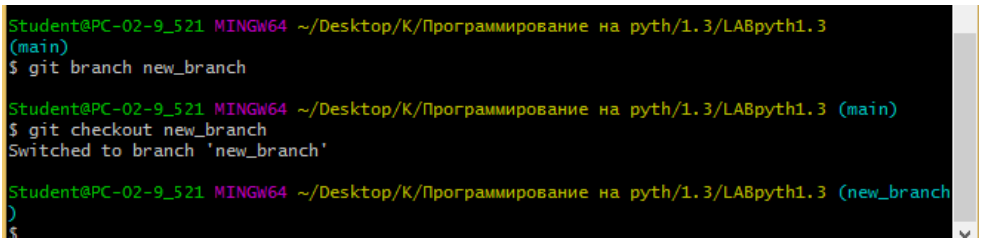

Рисунок 6 – создание новой ветки

После было успешно произведено сливание веток main с my_first_branch, main и new_branch, а my_first_branch с new_branch затем ветки my_first_branch с new_branch были удалены.

Были созданы новые ветки:

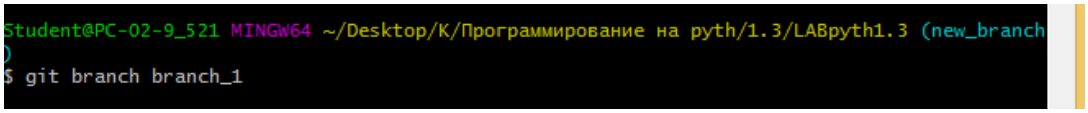

Рисунок 7 – branch_1

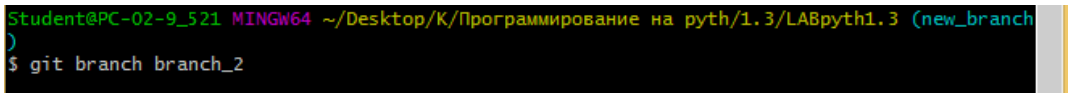

Рисунок 8 – branch_2

Далее, на первой ветке, были произведены следующие изменения:

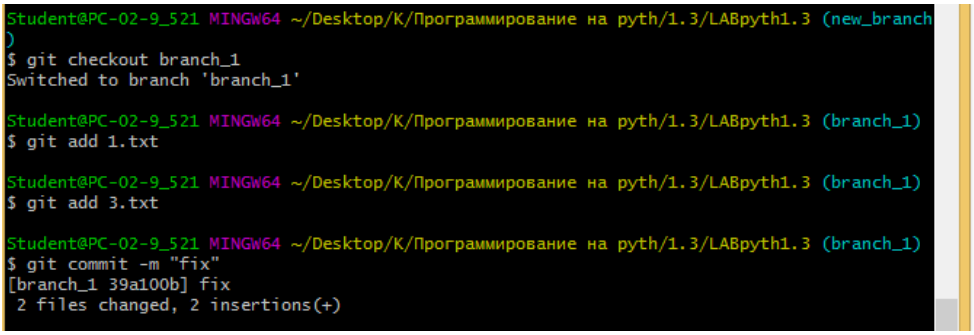

Рисунок 9 – изменения на первой ветке

Аналогично ветке 1 были сделаны изменения на ветке 2:

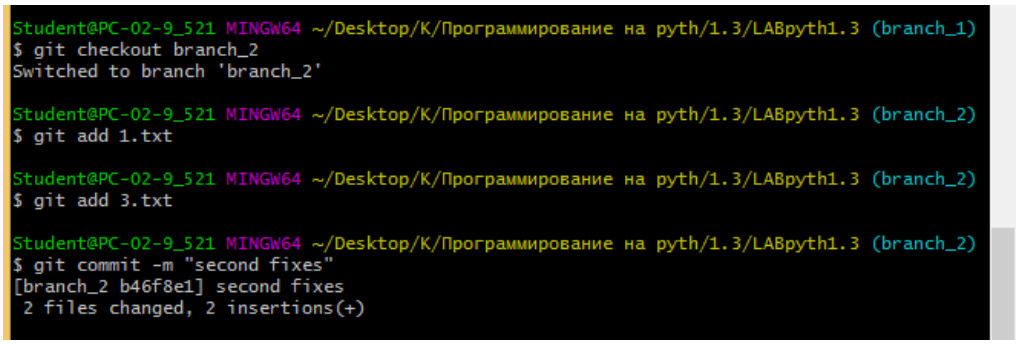

Рисунок 10 – изменения на второй ветке

Следующий этап – сливание этих двух веток.

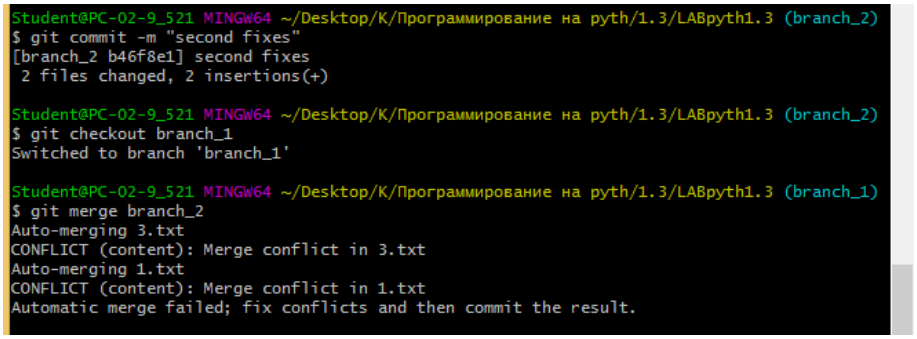

Рисунок 11 – сливание веток

На скрине видно, что сливание не закончилось, а осталось в режиме ожидания. Это произошло потому, что при сливании веток возник конфликт в файлах 1 и 3.

Ручное решение конфликта

Для этого способа необходимо самостоятельно зайти в файл и просмотреть его содержимое. Будет видно следующее:

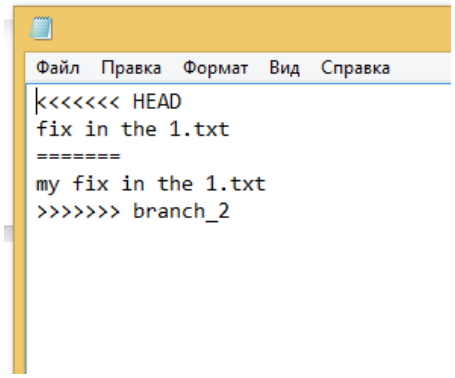

Рисунок 12 – конфликт в файле 1

Для решения этого конфликта нужно стереть все лишние строки и оставить те, которые необходимо слить.

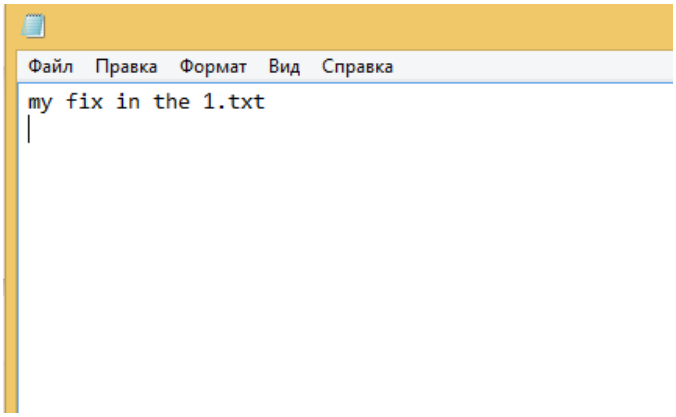

Рисунок 13 – решение конфликта в файле 1

Далее следует обозначить изменения в терминале, для чего используется обычная индексация.

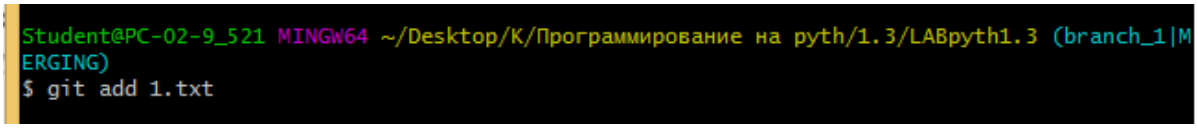

Рисунок 14 – индексация файла 1 после решения конфликта

Автоматическое решение конфликта

Для того, чтобы не делать лишних действий при решении конфликта, существует команда git margetool. После её ввода необходимо выбрать тип инструмента (утилиты), который сможет устранить конфликт самостоятельно.

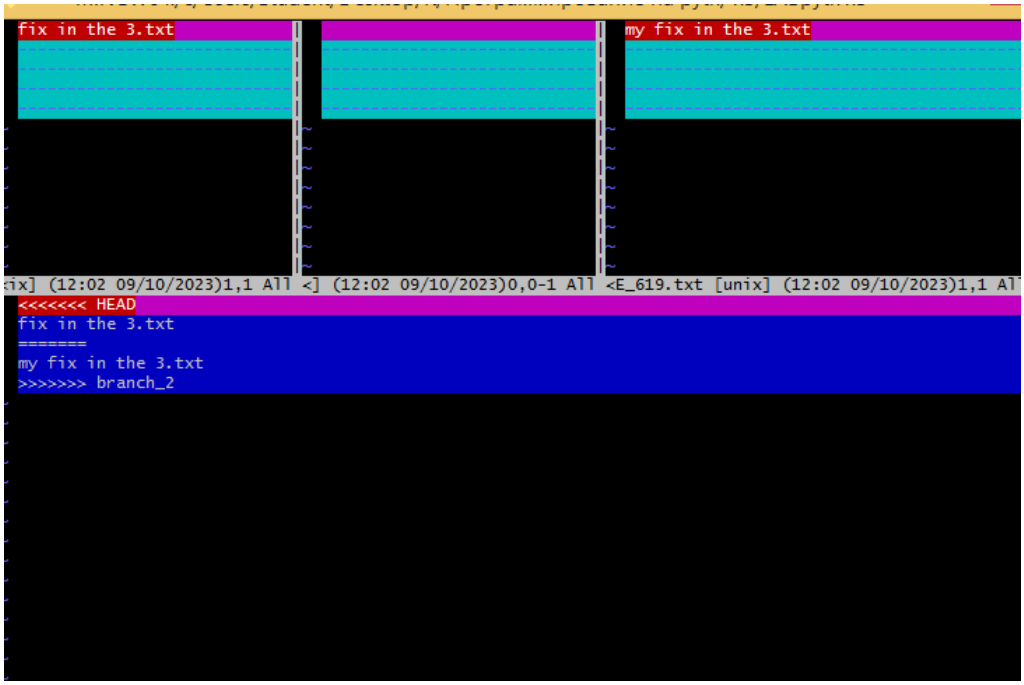

Рисунок 15 – git margetooll *meld*

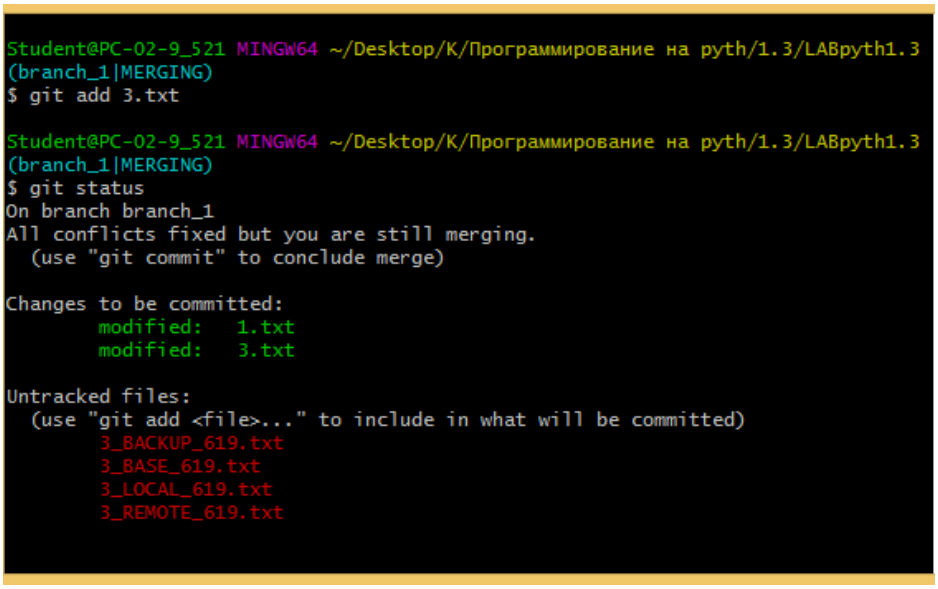

Рисунок 16 – индексация файла 3 после решения конфликта

После индексации всех файлов необходимо продолжить сливание.

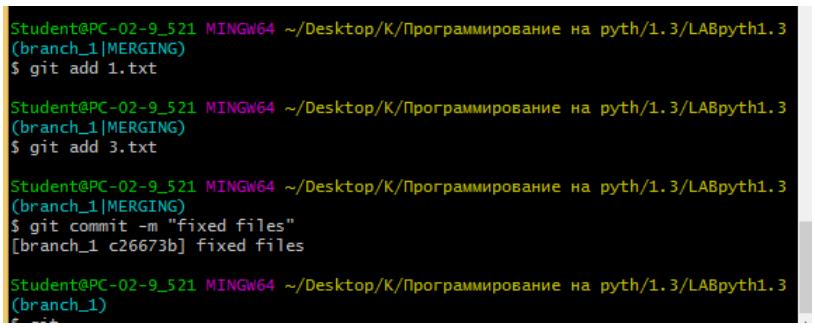

Рисунок 17 – конец сливания

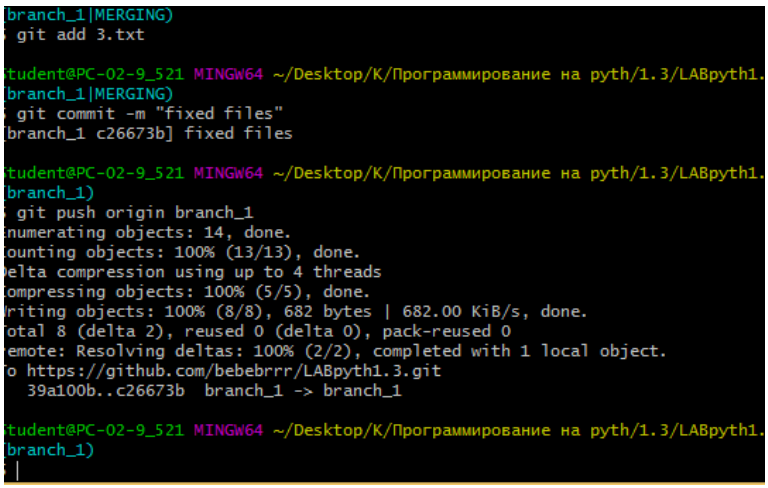

Рисунок 18 – отправка изменений

Работа с удалёнными ветками

Для демонстрации работы с удалёнными ветками необходимо для начала создать ветку в удалённом репозитории:

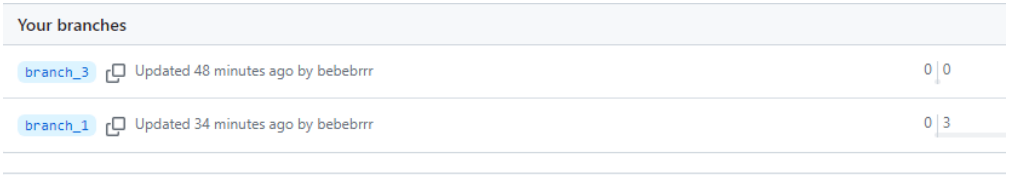

Рисунок 19 – ветка branch_3

Теперь, с помощью консоли, нужно связаться с этой веткой.

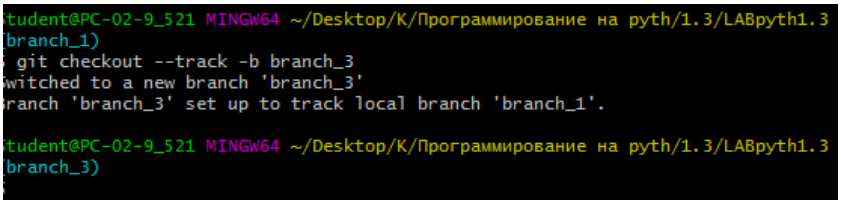

Рисунок 20 – отслеживание ветки

В данном случае HEAD автоматически перешло на отслеживаемую ветку с помощью опции -b.

Были созданы соответствующие изменения в файле 2, а он, в свою очередь, проиндексирован и закоммитен.

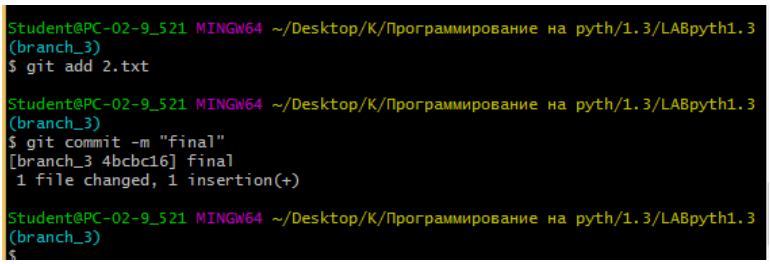

Рисунок 21 – индексирование файла 2

В заключении работы производится успешное сливание веток main и branch_2.

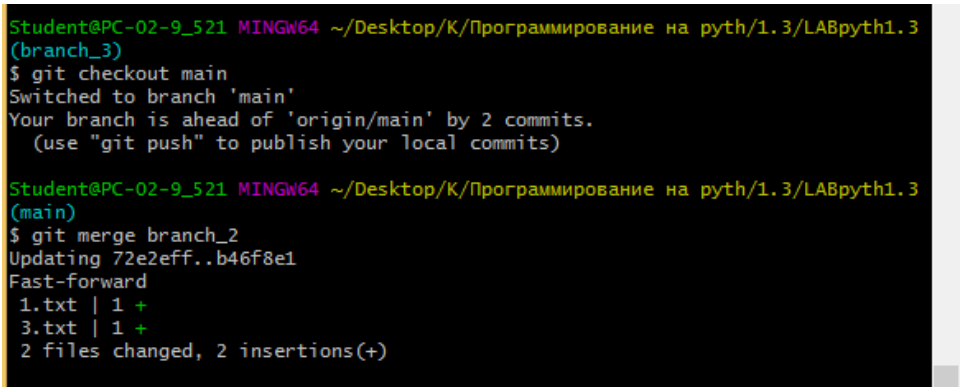

Рисунок 22 – сливание веток

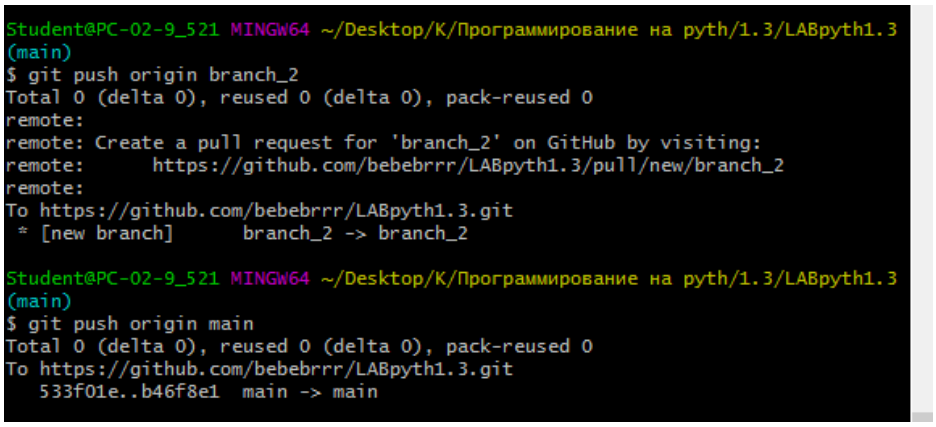

Рисунок 23 – отправка изменений

Контрольные вопросы

1. Что такое ветка?

Используя ветвление, Вы отклоняетесь от основной линии разработки и продолжаете работу независимо от неё, не вмешиваясь в основную линию. По сути говоря, ветка – это один из путей, по которым мы идём для внесения каких-либо изменений.

2. Что такое HEAD?

HEAD – это указатель, задача которого ссылаться на определенный коммит в репозитории.

3. Способы создания веток.

Ветку можно создать локально (в самом терминале) и удалённо.

4. Как узнать текущую ветку?

Git branch -a или git status.

5. Как переключаться между ветками?

Git checkout <branch>

6. Что такое удаленная ветка?

Ветка, расположенная на удалённом репозитории.

7. Что такое ветка отслеживания?

Локальная ветка для отслеживания изменений на удалённой ветке.

8. Как создать ветку отслеживания?

Git checkout –track <branch>

9. Как отправить изменения из локальной ветки в удаленную ветку?

Git push origin <branch>

10. В чем отличие команд git fetch и git pull ?

Обе команды получают обновления с удалённого репозитория, однако git fetch выполняет загрузку последний изменений, но не автоматически объединяет их с их текущей веткой. Изменения сохраняются в отдельной ветке, на которой они были записаны ранее. Git pull в этом смысле удобнее, поскольку он автоматически сразу сливает текущую ветку в локальном репозитории с удалённой веткой и всеми изменениями в ней.

11. Как удалить локальную и удаленную ветки?

Для удаления локально достаточно ввести команду git branch -d, а для удаления удалённой – git push origin –delete <branch>.

12. Какие основные типы веток присуствуют в модели git-flow? Как организована работа с ветками в модели git-flow? В чем недостатки git-flow?

В модели git-flow основными типами веток являются:

1) Master (Главная ветка): В этой ветке находится стабильный код, который всегда готов к развертыванию в продакшн.

2) Develop (Разрабатываемая ветка): В этой ветке происходит активная разработка функций и исправление ошибок. Она является основной веткой для интеграции всех фич в проект.

3) Feature (Фича-ветки): Каждая новая функция или фича разрабатывается в собственной ветке. Она создается от ветки Develop и после завершения работы интегрируется обратно в Develop при помощи merge или pull request.

4) Release (Ветки релизов): Когда разработка на ветке Develop достигает определенного состояния стабильности и готовности к релизу, создается ветка Release. В этой ветке выполняется тестирование, исправление ошибок и подготовка к финальному релизу.

5) Hotfix (Ветки исправлений): Если в продакшн версии обнаруживается критическая ошибка, создается ветка Hotfix. Она основана на ветке Master, позволяет внести исправления и затем объединяется обратно и в Master и в Develop.

Организация работы с ветками в git-flow основана на четком разделении ответственности между ветками и командами разработчиков. Каждая ветка имеет свою роль и правила интеграции.

Например, фича-ветки разрабатываются независимо друг от друга и интегрируются в Develop. Release-ветки позволяют подготовить стабильную версию перед релизом. Hotfix-ветки позволяют быстро исправить критические ошибки в продукции.

Недостатки git-flow:

1) Сложность: Git-flow может быть сложной моделью для понимания и использования, особенно для новых разработчиков. Она требует дополнительного обучения и понимания концепций модели.

2) Жесткость: Git-flow предлагает строгую организацию работы с ветками, что может быть связано с ограничениями и сложностями при изменении планов и разработке нестандартных фич.

3) Много веток: Git-flow может привести к большому количеству веток в репозитории, особенно если проект активно развивается и содержит много фич и релизов. Управление этими ветками может стать сложной задачей.

13. На **прошлой лабораторной работе было задание выбрать одно из программных средств с GUI для работы с Git. Необходимо в рамках этого вопроса привести описание инструментов для работы с ветками Git, предоставляемых этим средством.**

Codeberg представляет несколько инструментов для работы с ветками Git. Команда git branch, git checkout. А также веб-интерфейс Codeberg позволяет просматривать список веток и переключаться между ними с помощью интерфейса пользователя. Помимо этого, на Codeberg существуют Pull-запросы, с помощью которых можно сотрудничать и обсуждать изменения веток. С помощью веб-интерфейса удобно перемещаться и управлять ветками.

Вывод: базовые возможности по работе с локальными и удалёнными ветками в Git заключаются в создании веток, переключением между ними, а так же удалении веток.