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 «Основы программной инженерии»

Выполнил:
Мелтонян Одиссей
2 курс, группа ПИЖ-б-о-22-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка
и сопровождение программного
обеспечения», очная форма обучения

_____
(подпись)

Проверил: Воронкин Р. А.

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г

Тема: Основы ветвления Git

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

Ход работы:

1. Изучил теоретический материал работы.

2. Создал общедоступный репозиторий на GitHub, в котором была использована лицензия MIT.

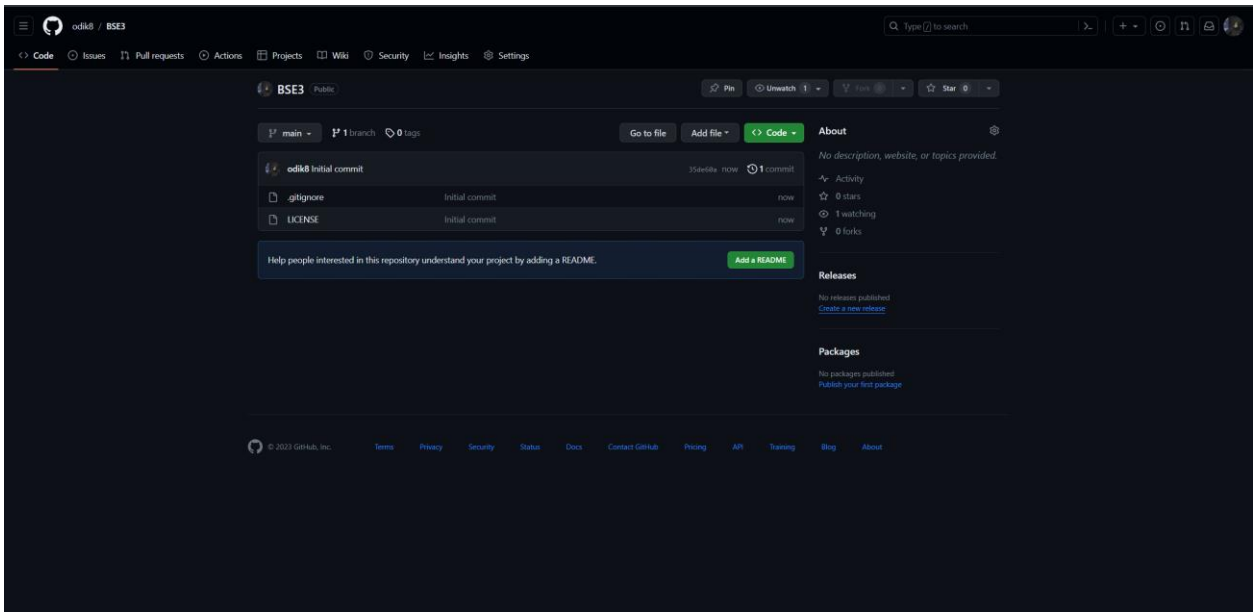

Рисунок 1. – Созданный репозиторий

3. Создал три файла: 1.txt, 2.txt, 3.txt.

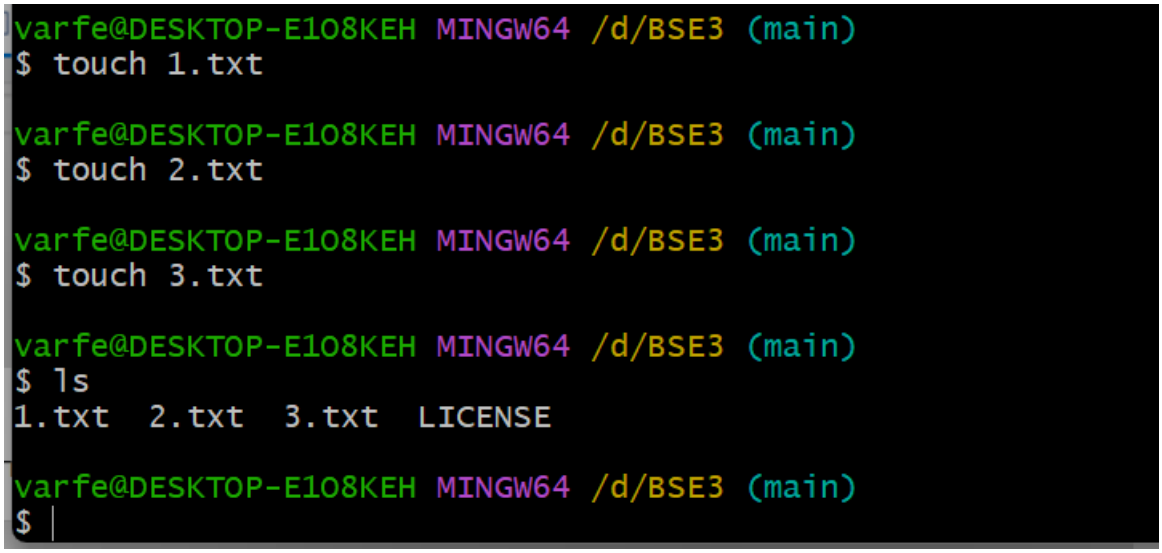

Рисунок 2. – Создание файлов

4. Проиндексировал первый файл и сделал коммит с комментарием "add 1.txt file".

```
varfe@DESKTOP-E1O8KEH MINGW64 /d/BSE3 (main)
$ git commit -m "add 1.txt file"
[main 1431397] add 1.txt file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 1.txt
```

Рисунок 3. – Индексация и коммит первого файла

5. Проиндексировал второй и третий файлы.

```
varfe@DESKTOP-E1O8KEH MINGW64 /d/BSE3 (main)
$ git add 2.txt

varfe@DESKTOP-E1O8KEH MINGW64 /d/BSE3 (main)
$ git add 3.txt

varfe@DESKTOP-E1O8KEH MINGW64 /d/BSE3 (main)
$
```

Рисунок 4. – Индексация остальных фалов

6. Перезаписал уже сделанный коммит с новым комментарием "add 2.txt and 3.txt."

```
MINGW64:/d/BSE3
add 2.txt and 3.txt

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Tue Oct 24 21:46:45 2023 +0300
#
# On branch main
# Your branch is ahead of 'origin/main' by 1 commit.
#   (use "git push" to publish your local commits)
#
# Changes to be committed:
#       new file:   1.txt
#       new file:   2.txt
#       new file:   3.txt
#
~
```

Рисунок 5. – Перезапись коммита

7. Создал новую ветку my_first_branch

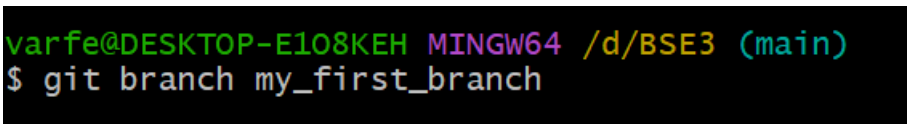

Рисунок 6. – Создание новой ветки

8. Перешел на ветку и создал новый файл in_branch.txt, закоммитить изменения

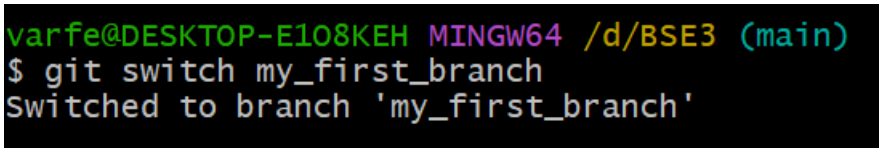

Рисунок 7. – Переход на новую ветку

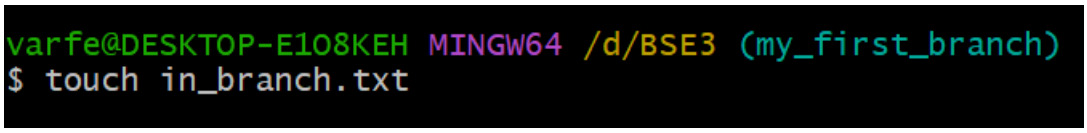

Рисунок 8. – Создание нового файла

9. Вернулся на ветку main

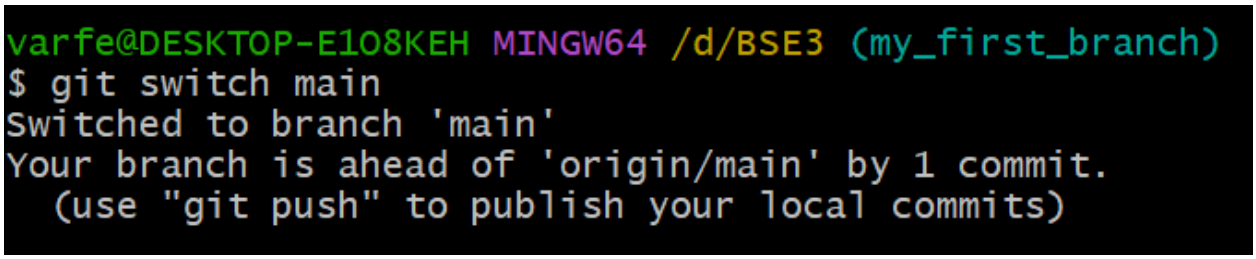

Рисунок 9. – Возврат на ветку main

10. Создал и сразу перешел на ветку new_branch

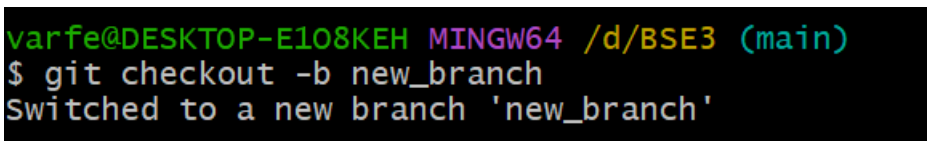

Рисунок 10. – Создание и переход на новую ветку

11. Сделал изменения в файле 1.txt, добавил строчку “new row in the 1.txt file”, закоммитил изменения.

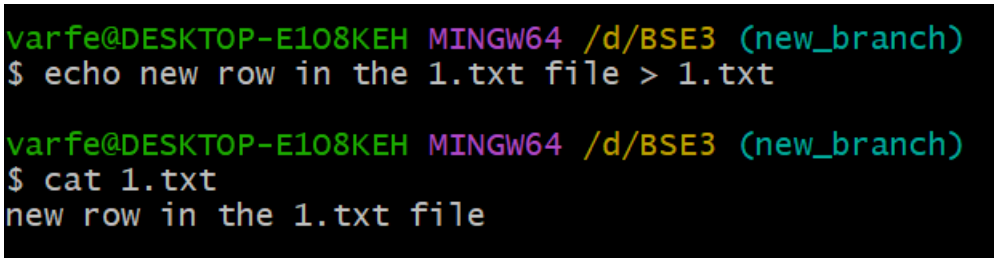

Рисунок 11. – Изменение файла 1.txt

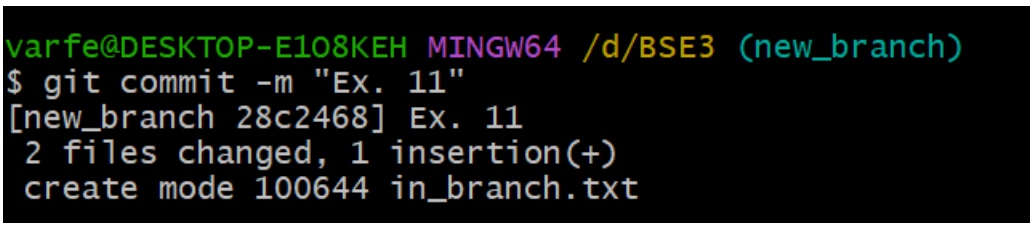

Рисунок 12. – Коммит изменений

12. Перешел на ветку master и слил ветки master и my_first_branch, после чего слил ветки master и new_branch.

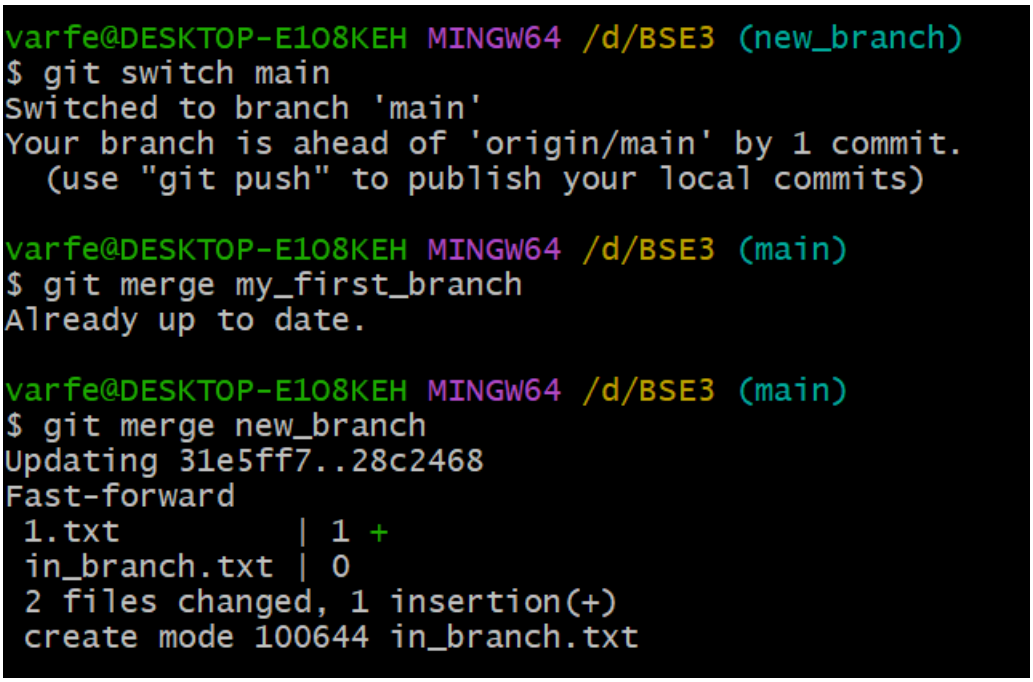

Рисунок 13. – Слияние веток

13. Удалил ветки my_first_branch и new_branch.

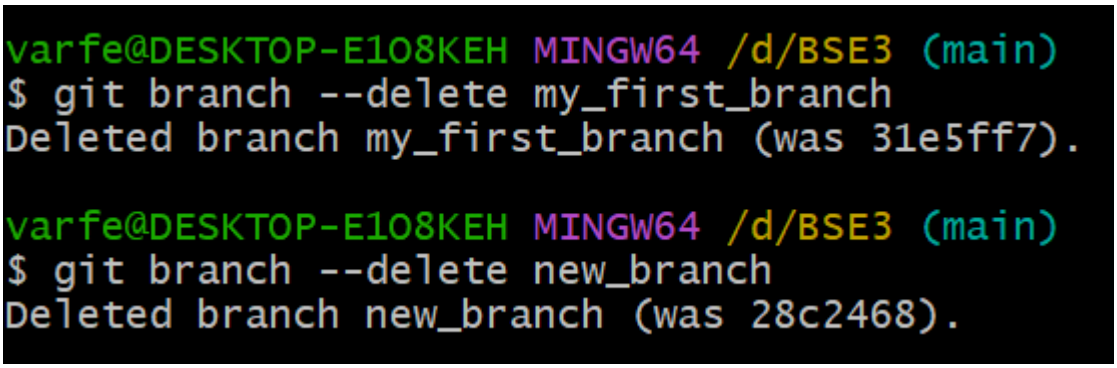

Рисунок 14. – Удаление веток

14. Создал ветки branch_1 и branch_2.

```
varfe@DESKTOP-E1O8KEH MINGW64 /d/BSE3 (main)
$ git branch branch_1

varfe@DESKTOP-E1O8KEH MINGW64 /d/BSE3 (main)
$ git branch branch_2
```

Рисунок 15. – Создание новых веток

15. Перейти на ветку branch_1 и изменить файл 1.txt, удалить все содержимое и добавить текст “fix in the 1.txt”, изменить файл 3.txt, удалить все содержимое и добавить текст “fix in the 3.txt”, закоммитил изменения.

```
varfe@DESKTOP-E1O8KEH MINGW64 /d/BSE3 (main)
$ git switch branch_1
Switched to branch 'branch_1'
```

Рисунок 16. – Переход на ветку branch_1

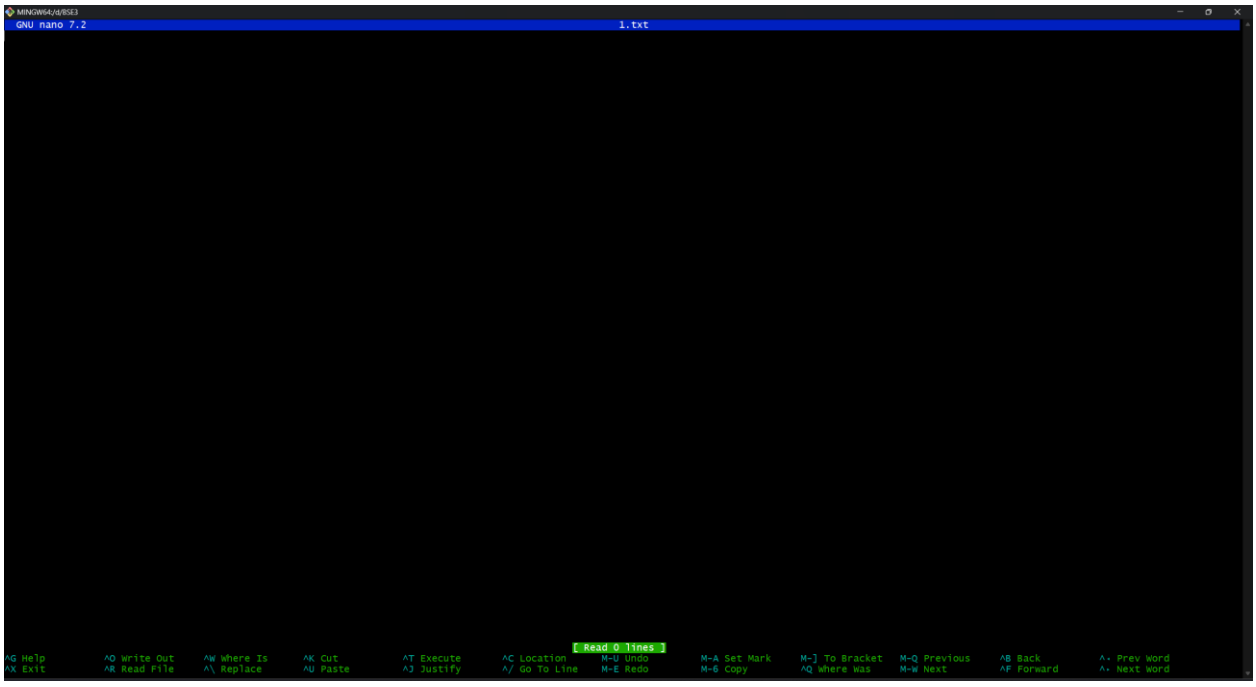

Рисунок 17. – Удаление содержимого файла 1.txt

```
varfe@DESKTOP-E1O8KEH MINGW64 /d/BSE3 (branch_1)
$ echo "fix in the 1.txt" > 1.txt
```

Рисунок 18. – Добавление текста в 1.txt

```
varfe@DESKTOP-E1O8KEH MINGW64 /d/BSE3 (branch_1)
$ echo "fix in the 3.txt" > 3.txt
```

Рисунок 19. – Добавление текста в 3.txt

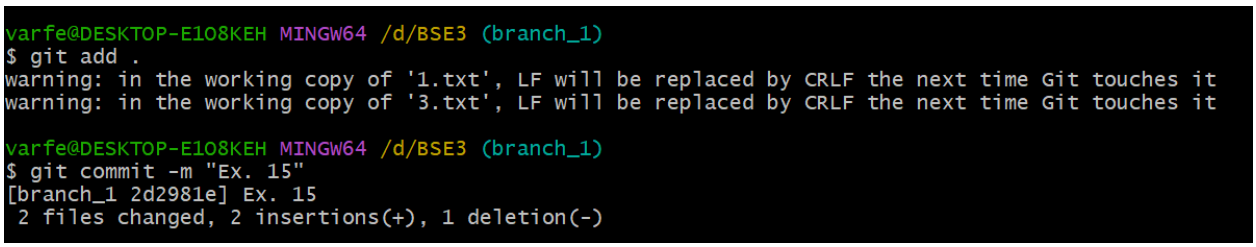

Рисунок 20. – Коммит изменений

16. Перешел на ветку branch_2 и также изменил файл 1.txt, удалил все содержимое и добавил текст “My fix in the 1.txt”, изменил файл 3.txt, удалил все содержимое и добавил текст “My fix in the 3.txt”, закоммитил изменения.

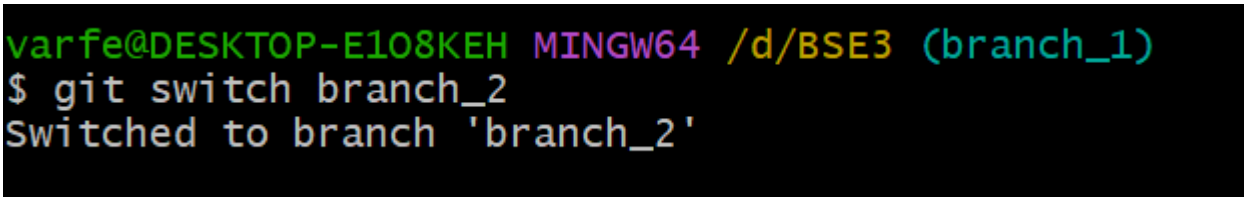

Рисунок 21. – Переход на ветку branch_2

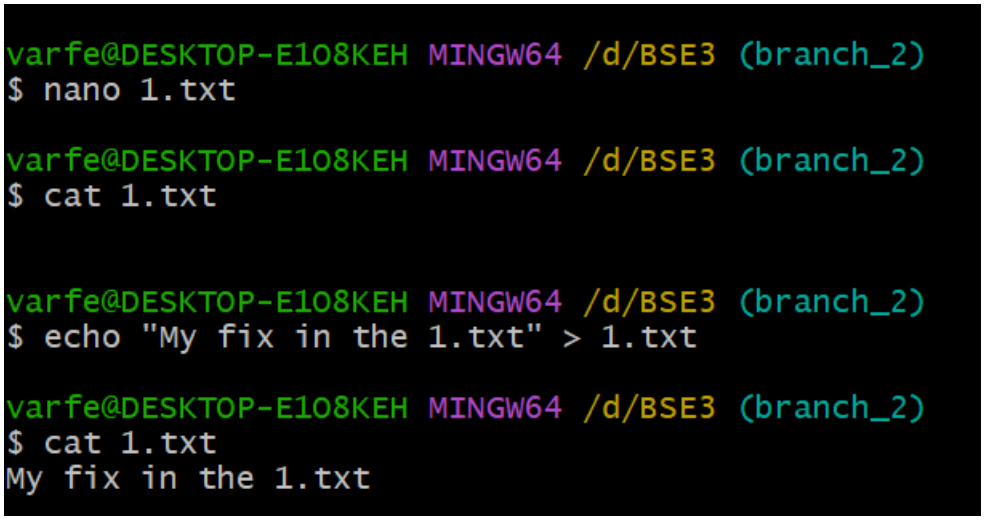

Рисунок 22. – Изменение содержимого файла 1.txt

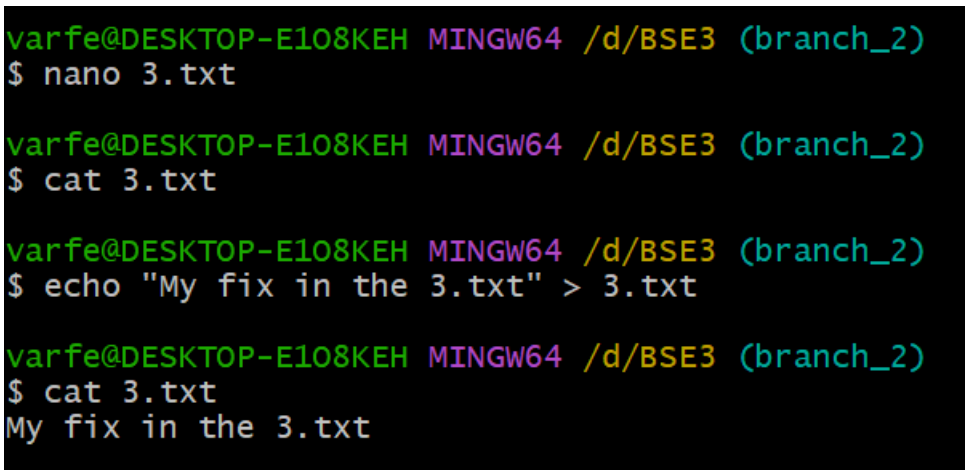

Рисунок 23. – Изменение содержимого файла 3.txt

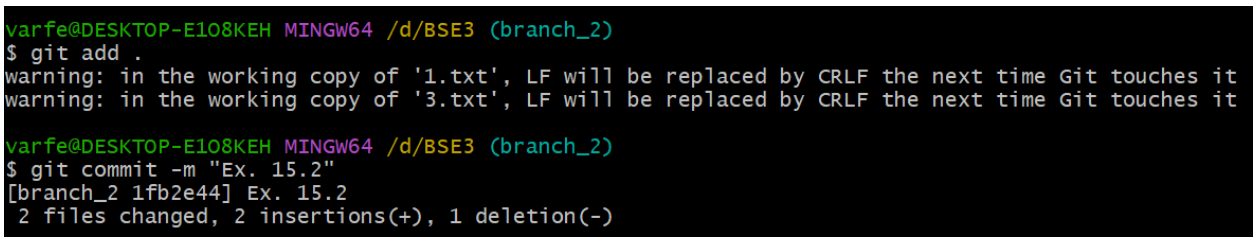

Рисунок 24. – Коммит изменений

17. Слить изменения ветки branch_2 в ветку branch_1.

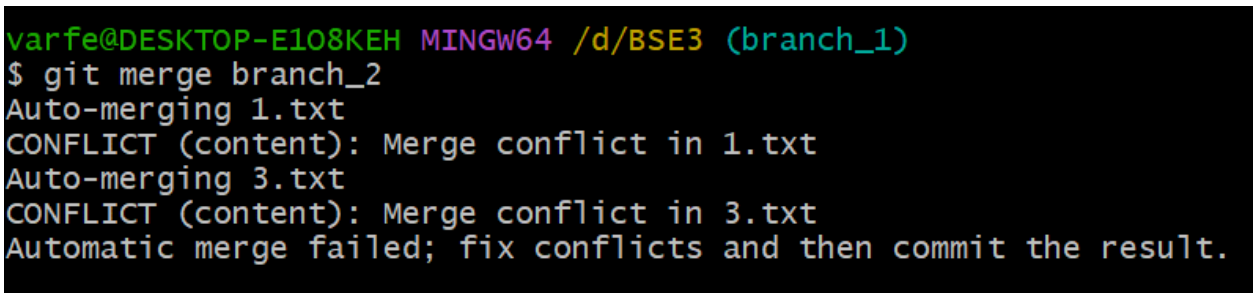

Рисунок 25. – Конфликт при слиянии веток

18. Решил конфликт файла 1.txt в ручном режиме, а конфликт 3.txt используя команду git mergetool с помощью одной из доступных утилит, например Meld.

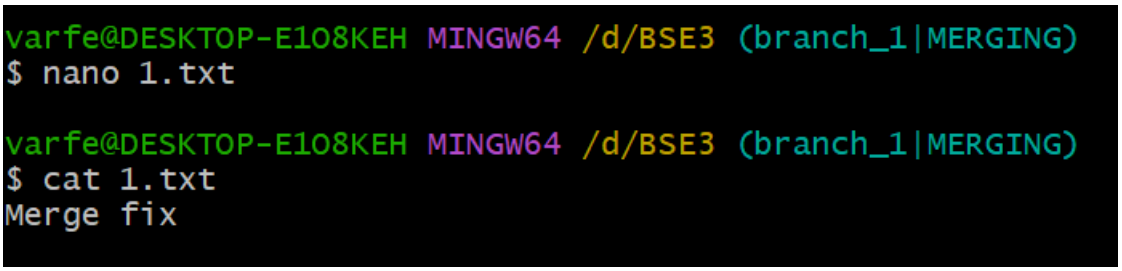

Рисунок 26. – Решение конфликта вручную

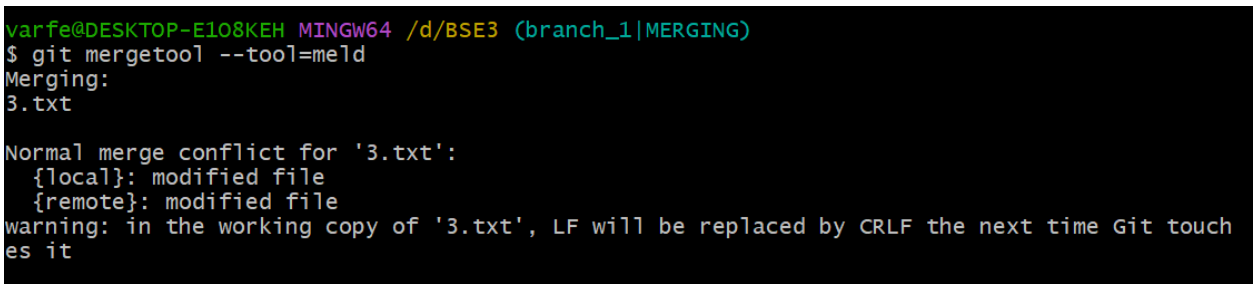

Рисунок 27. – Открытие утилиты Meld

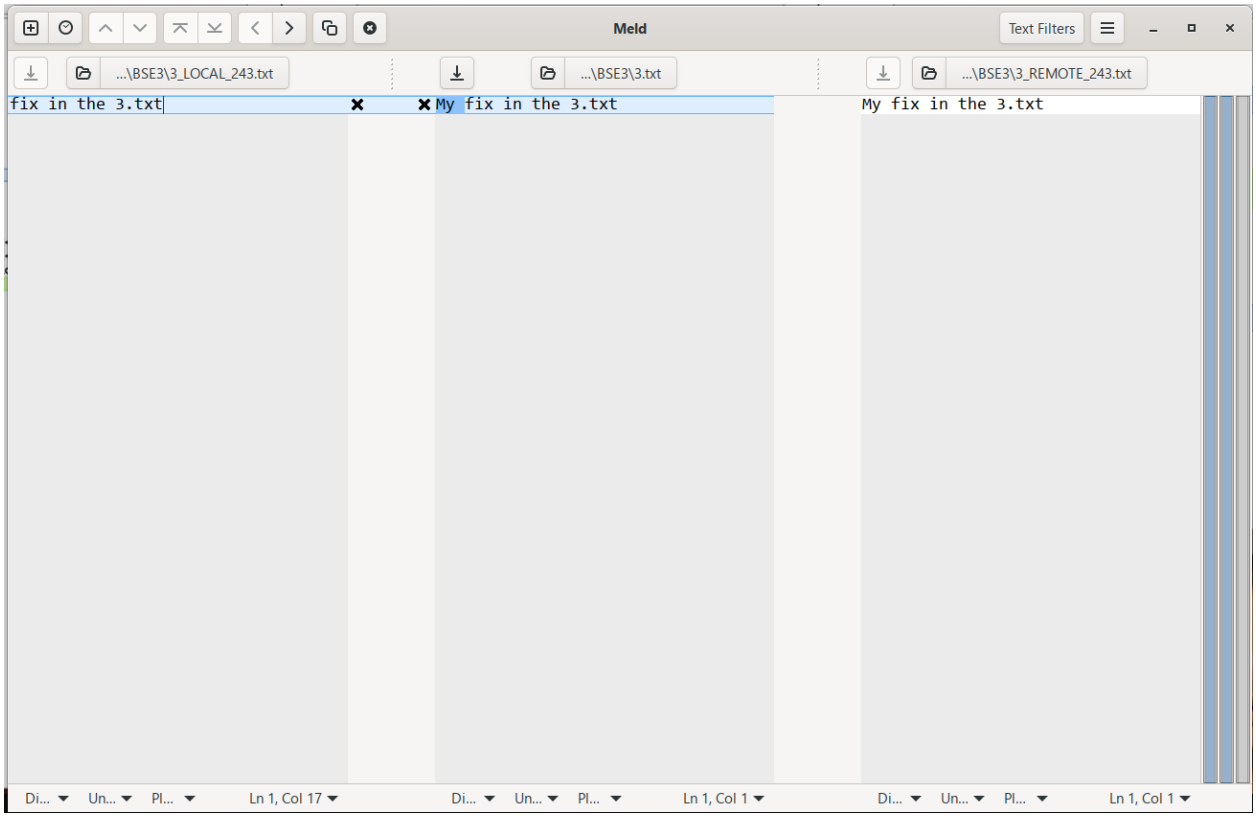

Рисунок 28. – Решение конфликта с помощью утилиты Meld

19. Отправил ветку branch_1 на GitHub

```
varfe@DESKTOP-E1O8KEH MINGW64 /d/BSE3 (branch_1)
$ git push --set-upstream origin branch_1
Enumerating objects: 18, done.
Counting objects: 100% (18/18), done.
Delta compression using up to 6 threads
Compressing objects: 100% (10/10), done.
Writing objects: 100% (17/17), 1.33 KiB | 1.33 MiB/s, done.
Total 17 (delta 5), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (5/5), done.
remote:
remote: Create a pull request for 'branch_1' on GitHub by visiting:
remote:      https://github.com/odik8/BSE3/pull/new/branch_1
remote:
To https://github.com/odik8/BSE3.git
 * [new branch]      branch_1 -> branch_1
branch 'branch_1' set up to track 'origin/branch_1'.
```

Рисунок 29. – Пуш ветки

20. Создал средствами GitHub удаленную ветку branch_3

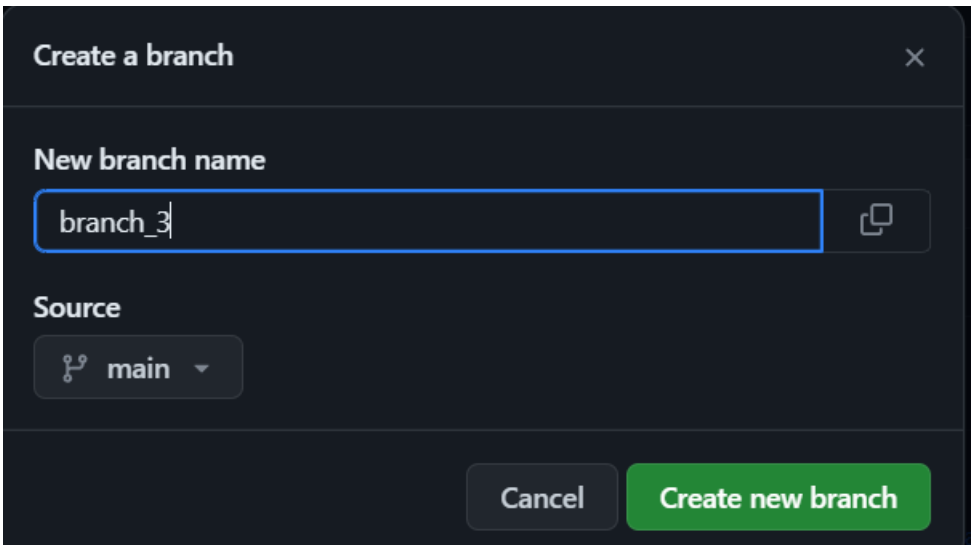

Рисунок 30. – Создание ветки branch_3

21. Создал в локальном репозитории ветку отслеживания удаленной ветки branch_3.

```
varfe@DESKTOP-E1O8KEH MINGW64 /d/BSE3 (branch_2)
$ git checkout --track origin/branch_3
Switched to a new branch 'branch_3'
M       2.txt
branch 'branch_3' set up to track 'origin/branch_3'.
```

Рисунок 31. – Создание ветки отслеживания

22. Перешел на ветку branch_3 и добавить в файл 2.txt строку "the final fantasy in the 4.txt file" (с помощью команды nano).

```
varfe@DESKTOP-E1O8KEH MINGW64 /d/BSE3 (origin/branch_3)
$ nano 2.txt

varfe@DESKTOP-E1O8KEH MINGW64 /d/BSE3 (origin/branch_3)
$ cat 2.txt
the final fantasy in the 4.txt file
```

Рисунок 32. – Добавление строки

23. Выполнил перемещение ветки main на ветку branch_2.

Рисунок 33.

24. Отправил изменения веток main и branch_2 на GitHub

Рисунок 34. – Отправка изменений

Вывод: были исследованы базовые возможности по работе с локальными и удаленными ветками Git.

Вопросы для защиты работы

1. Что такое ветка? – В контексте системы контроля версий Git, ветка (branch) представляет собой отдельную линию разработки, которая содержит

набор коммитов, представляющих определенную последовательность изменений в проекте. Ветки позволяют разработчикам работать параллельно над разными функциональными или экспериментальными частями проекта.

2. Что такое HEAD? – это специальный указатель в Git, который указывает на текущий коммит или текущую ветку. HEAD используется для определения текущего состояния репозитория. Если HEAD указывает на ветку, это означает, что вы в данный момент работаете с этой веткой.

3. Способы создания веток.

- **git branch <имя_ветки>**: создаёт новую ветку, но не переключает на нее.
- **git checkout -b <имя_ветки>**: создаёт новую ветку и переключает на нее.
- **git switch -c <имя_ветки>**: аналогично создает и переключает на новую ветку.
- **git clone <репозиторий>**: создаёт локальную копию удаленного репозитория, включая его ветки.

4. Как узнать текущую ветку? – Чтобы узнать текущую ветку, выполните команду **git branch**. Текущая ветка будет отмечена символом "*".

5. Как переключаться между ветками? – Для переключения между ветками используйте команду **git checkout <имя_ветки>**, либо **git switch <имя_ветки>** начиная с Git версии 2.23.

6. Что такое удаленная ветка? – Удаленная ветка (remote branch) - это ветка, которая существует в удаленном репозитории (например, на GitHub), а не локально на вашем компьютере. Она обновляется при синхронизации с удаленным репозиторием.

7. Что такое ветка отслеживания? – Ветка отслеживания (tracking branch) – это локальная ветка, которая связана с удаленной веткой. Она автоматически отслеживает изменения удаленной ветки, позволяя вам легко синхронизировать свои изменения с удаленным репозиторием.

8. Как создать ветку отслеживания? – Для создания ветки отслеживания выполните команду **git checkout --track <удаленный_репозиторий>/<удаленная_ветка>** или **git switch --track <удаленный_репозиторий>/<удаленная_ветка>**.

9. Как отправить изменения из локальной ветки в удаленную ветку? – Для отправки изменений из локальной ветки в удаленную ветку используйте команду **git push <удаленный_репозиторий> <локальная_ветка> : <удаленная_ветка>**.

10. В чем отличие команд git fetch и git pull? – Разница между **git fetch** и **git pull** заключается в том, что **git fetch** извлекает изменения из удаленного репозитория, но не объединяет их с текущей веткой, тогда как **git pull** извлекает и объединяет изменения.

11. Как удалить локальную и удаленную ветки? – Для удаления локальной ветки используйте **git branch -d <имя_ветки>**. Для удаления удаленной ветки используйте **git push origin --delete <branchName>**.

12. Какие основные типы веток присуствуют в модели git-flow? Как организована работа с ветками в модели git-flow? В чем недостатки git-flow? – Модель ветвления Git-Flow предлагает следующие типы веток:

- **main**: Главная ветка, в которой хранятся стабильные версии продукта.
- **develop**: Ветка разработки, в которой ведется активная разработка новых функций.
- **feature/**: Ветки для разработки новых функций.
- **release/**: Ветки для подготовки релизов.
- **hotfix/**: Ветки для исправления критических ошибок.

Работа с ветками в модели Git-Flow организована в соответствии с определенными правилами и сценариями, чтобы обеспечить удобное управление версиями продукта.

Недостатки Git-Flow включают сложность и необходимость внимательного соблюдения правил, что может быть избыточным для небольших проектов.