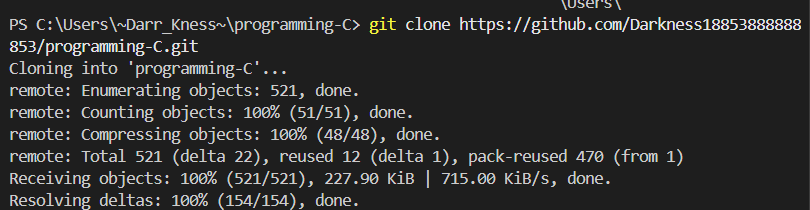
**Практическое задание 3. Git**

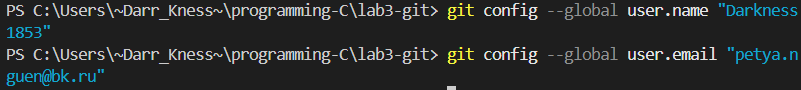
1)Клонируем репозиторий к себе.



2)Создаем в репозитории папку для лабы



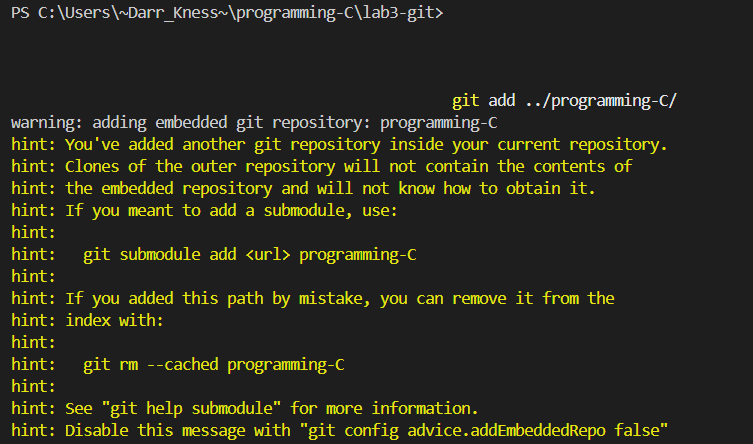
3)Логинимся

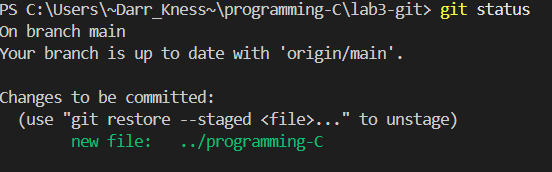
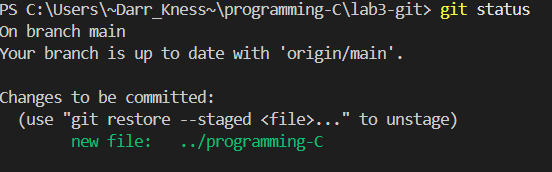


4)Выбираем текстовый редактор

На оценку 3:

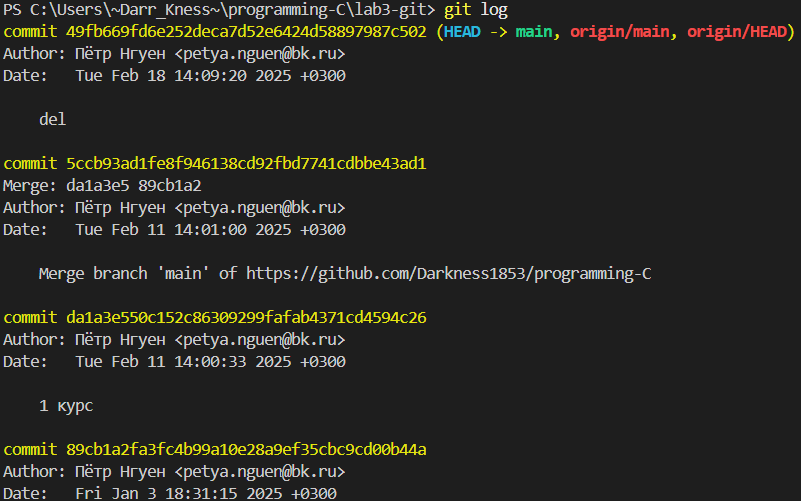
Добавляем папку в индекс(staging area) , чтобы Git начал отслеживать изменения в ней.

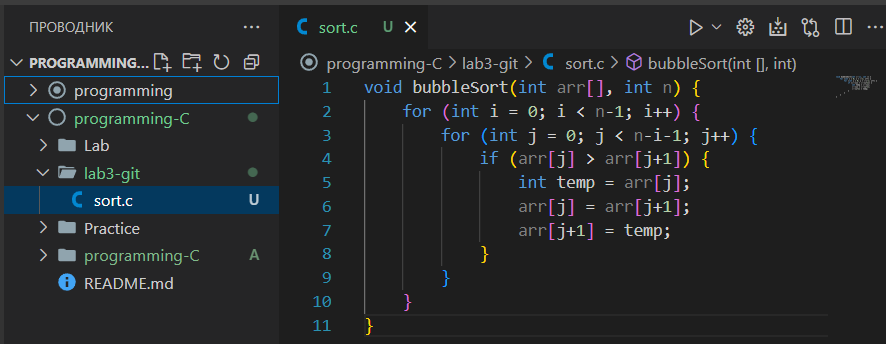
1)Используемgit status, чтобы узнать, на какой ветке мы находимся.

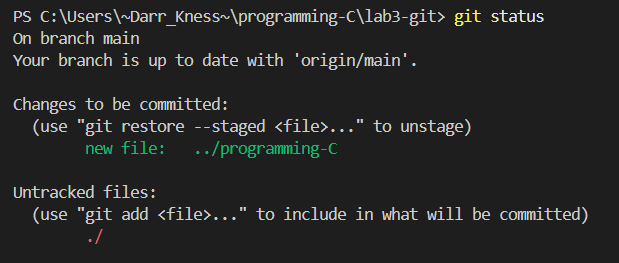
2)С помощью git log просмотрели истории коммитов в репозитории. Эта команда показывает список всех коммитов, начиная с последнего (самого свежего) и заканчивая самым первым коммитом в репозитории.

**Хэш коммита** (commit hash) — уникальный идентификатор коммита

1. ****Автор****(Author) — имя и email автора коммита.
2. **Дата** (Date) — дата и время создания коммита.
3. **Сообщение коммита** (commit message) — текст, который был указан при создании коммита

3)Создаем файл sort.c и вставляем код функции сортировки.

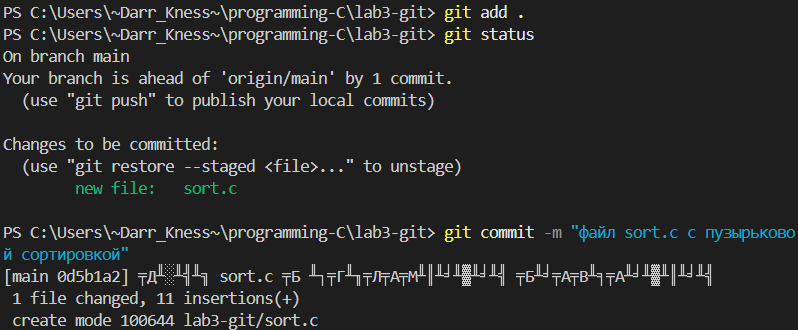
4)Вывод git status показало, что моей рабочей директории есть неотслеживаемые файлы. Эти файлы ещё не добавлены в staging area. Git предлагает использовать команду git add <file>, чтобы начать отслеживать их и включить в следующий коммит.

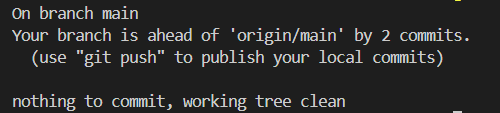
5)Добавим файл в область stage (add)

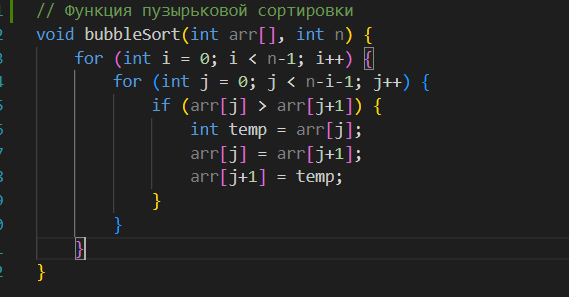
Добавим неотслеживаемые файлы в staging area:git add .(. — добавим все неизвестные файлы)

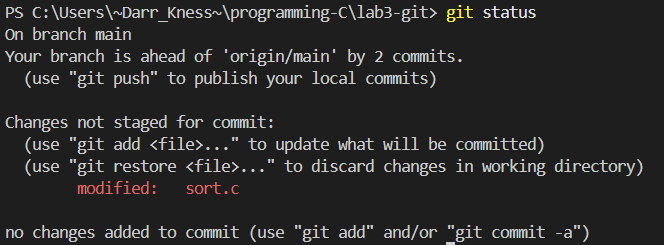
Проверьте статус снова: git status

Создайте коммит: git commit -m "файл sort.c с пузырьковой сортировкой"

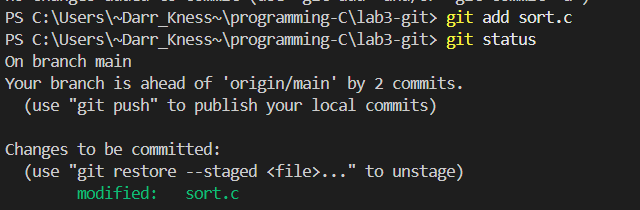
6)Проверим статус:

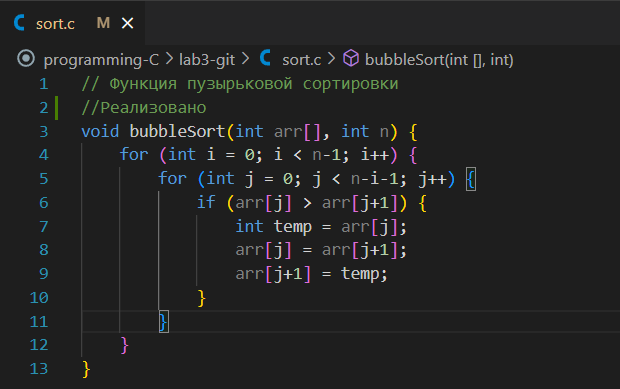
7)Добавили комментарий в созданный файл

10)Вот как выглядит git status

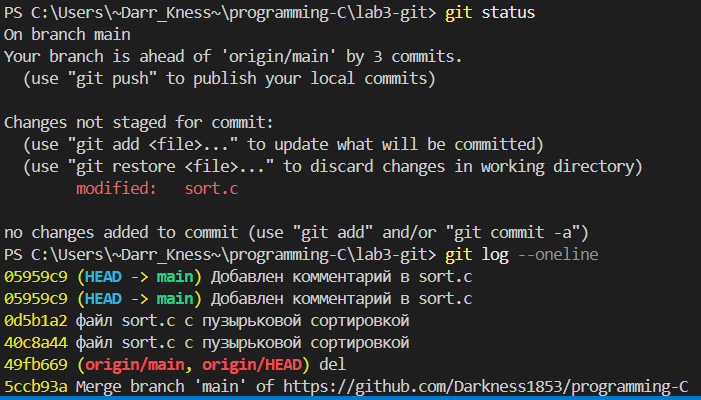
11)Добавим (add) изменение файла и

12)вызовем git status

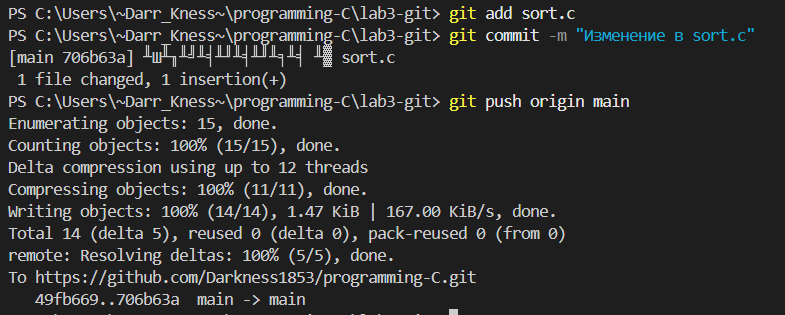
13)Изменим ещё раз файл

14)Проверим статус git status

15)И журнал с помощью команд git log --oneline

16) Добавим в stage и закоммитим последнее изменение

17) Запушим на удаленный репо (git push)

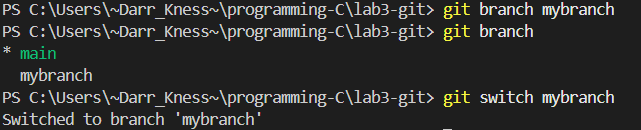


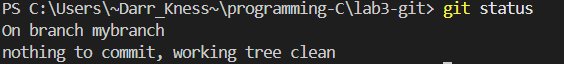
Работа с ветками

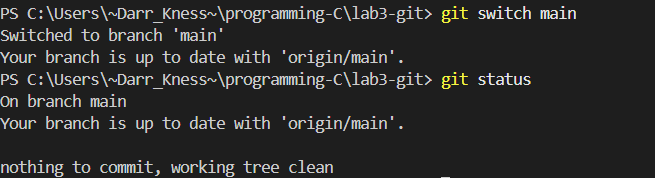
1)Создадим новую ветку с помощью команды git branch mybranch

2)Проверим список git branch

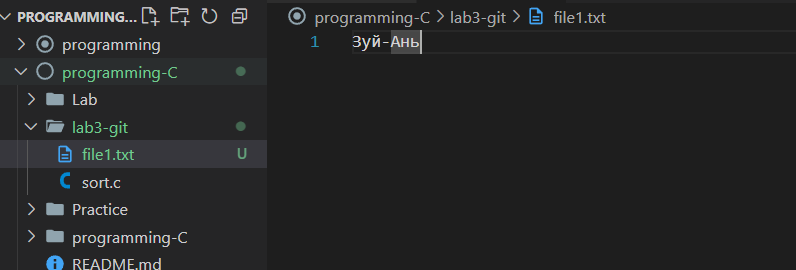
3)Переключимся на новую ветку

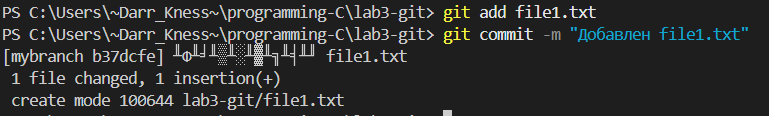
4)Git работает с ветками. В ветке mybranch нет изменений, и рабочая директория чиста, также как и в ветке main (нет изменённых или неотслеживаемых файлов).

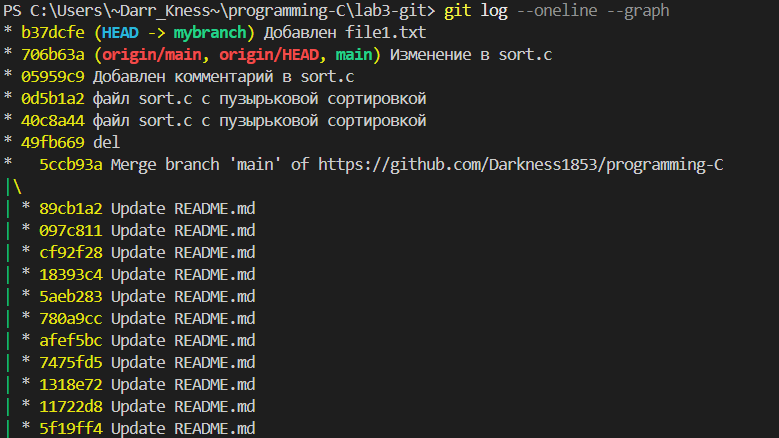


5)Вернулись в ветку mybranch

6)Создадим файл file1.txt и добавим в него свое имя.

7)Добавим файл и закоммитим это изменение.

8)Просмотрим журнал

9)Вернемся к ветке с именем master.

10)Используем git log --oneline —graph

Что изменилось:

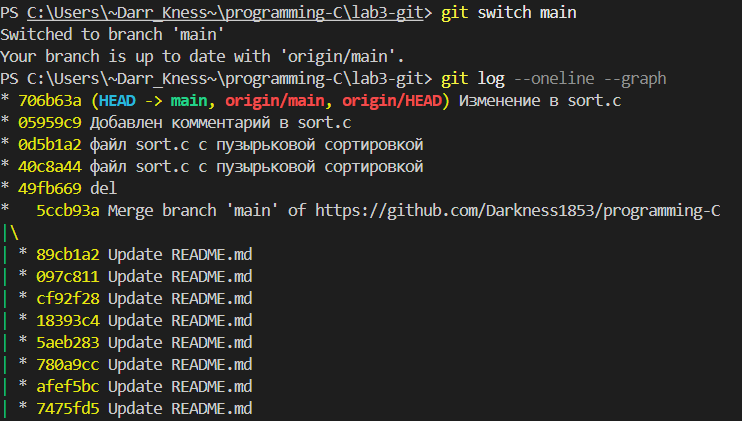
В первом выводе HEAD указывал на ветку mybranch, а во втором — на main.

В mybranch последний коммит — b37dcfe (добавление file1.txt).

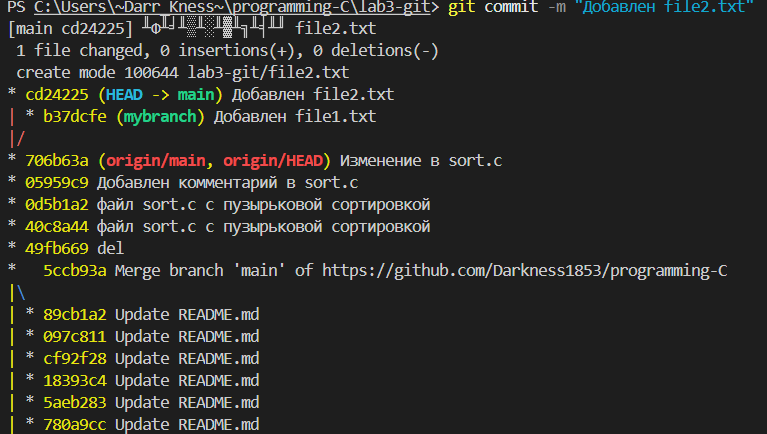
В main последний коммит — 706b63a (изменение в sort.c).

Коммит b37dcfe виден только в ветке mybranch, так как он был сделан в этой ветке и ещё не был слит с main.

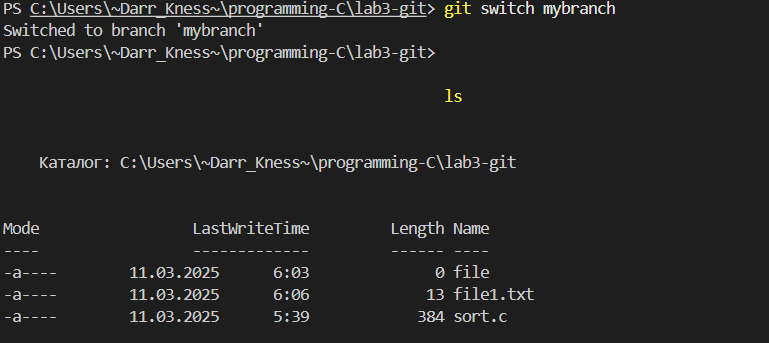
### Граф истории остался практически одинаковым, за исключением того, что в mybranch добавлен один дополнительный коммит (b37dcfe).

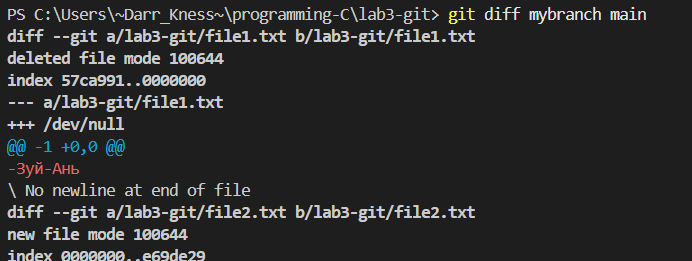
11)Создадим новый файл с именем file2.txt и закоммитим его.

12)Используя git log --oneline --graph –all, увидели, что моя ветка указывает на новый коммит, и что теперь у двух веток разные коммиты.

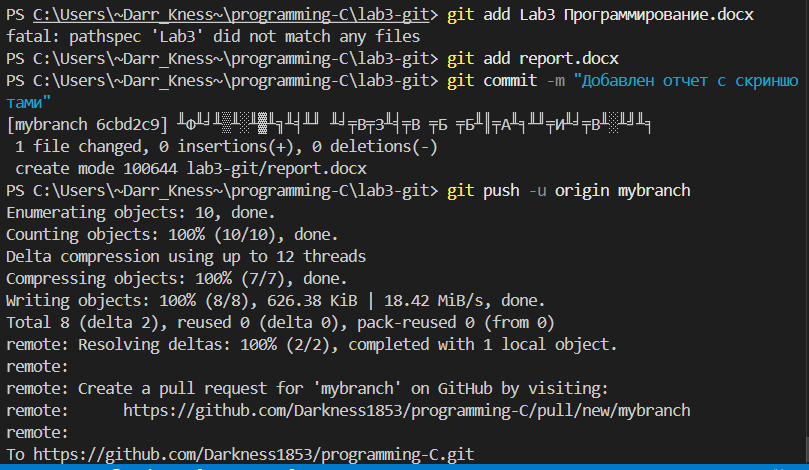
13)Переключимся на мою ветку mybranch.

14)Наш файл file2.txt пропал

15)Используя git diff mybranch main Сравним обе ветки

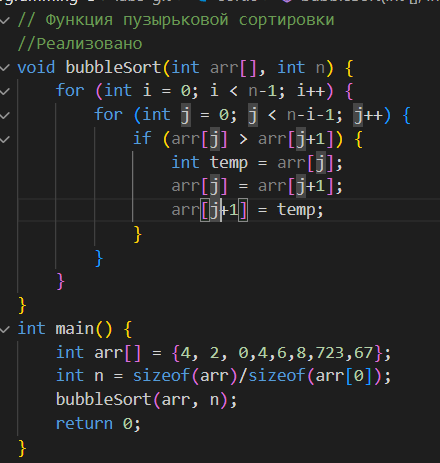
16)Добавим текстовый документ со скриншотами в ветку mybranch. Закоммитим и запушим на удаленный репозиторий ветку mybranch (git push -u origin mybranch)

Убедимься что в github.com две ветки master и mybranch. Не забудем запушить изменения master ветки в main

**На оценку 4:**

1)Переключимся в ветку mybranch

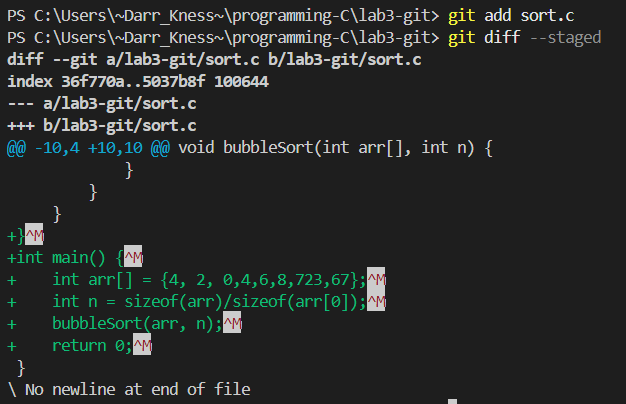
2)Перезапишем содержимое в sort.c добавив функцию main(), в которой будет объявлен массив из нескольких чисел (пример int a[] = {4, 2, 0};) и вызовем функция сортировки для этого массива.



3)Вывод команды git diff показывает изменения, которые были сделаны в файле sort.c, но ещё не были добавлены в staging area (индекс) с помощью git add.

Команда git diff сравнивает текущее состояние рабочей директории с последним закоммиченным состоянием (или состоянием в staging area, если файл был добавлен туда).в файл были добавлены новые строки. Они обозначены символом + (новые строки) и ^M (символ возврата каретки)



4)Команда git diff --staged показывает изменения, которые были добавлены в **staging area** (индекс) с помощью git add, но ещё не закоммичены.

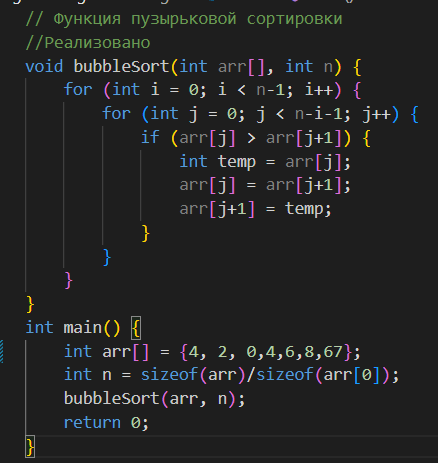
вывод команды **не пустой**, и он показывает, что файл lab3-git/sort.c был изменён и добавлен в staging area.

5)Добавили в staged файл sort.c

6) git diff выводит, что нет **изменений** в файле

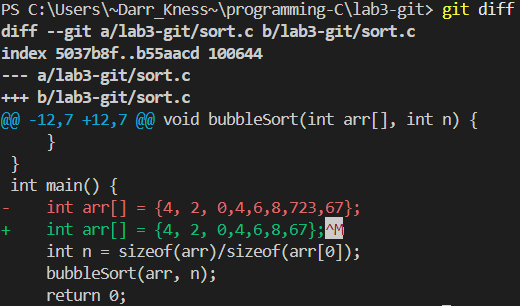
7) git diff —staged выводит, что в staging area находится изменённая версия файла sort.c

8)Удалим одно число из массива в sort.c



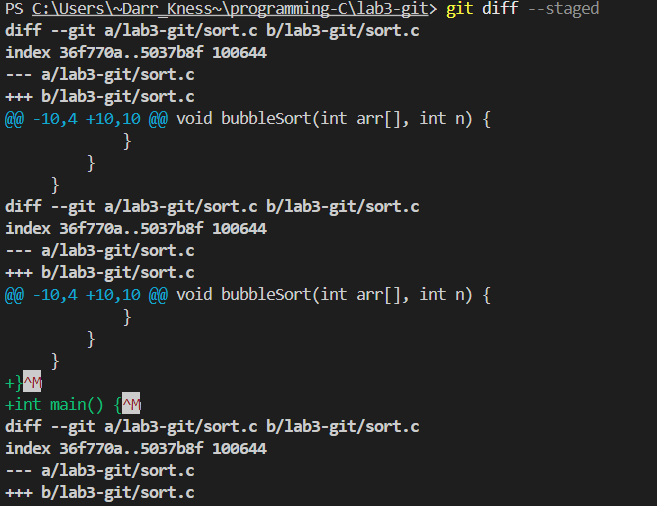
9)Проверим изменения с помощью команды git diff

Видим, что в файле была изменена строка, содержащая массив arr в функции main()



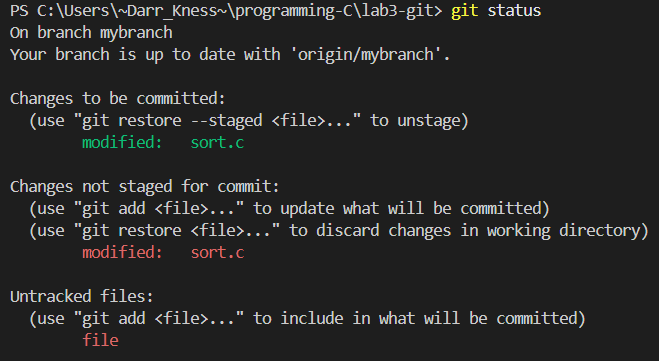
10)Выполним git diff —staged

В файл была добавлена новая функция main()

11)Обьясним, что происходит:Команда git diff --staged показывает изменения, которые были добавлены в **staging area** (индекс) с помощью git add, но ещё не закоммичены.

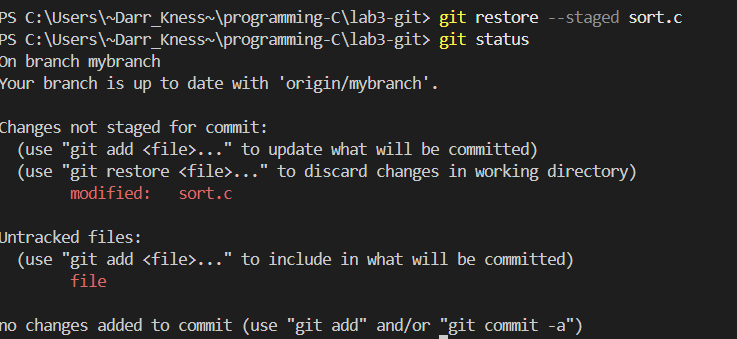
В файл была добавлена новая функция main()

12)Запустили git status и обратили внимание, что sort.c присутствует дважды в выводе.

13)Запустим git restore --staged sort.c , чтобы отменить индексацию изменения

Команда git restore --staged убирает файл из staging area (индекса), но **сохраняет изменения в рабочей директории**.

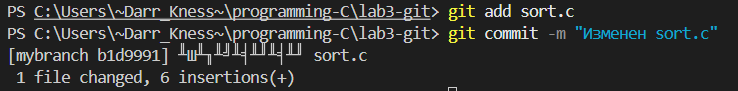
* После выполнения команды файл sort.c больше не готов к коммиту, но изменения в нём остаются.

14)Файл sort.c всё ещё изменён в рабочей директории, но эти изменения не добавлены в staging area.

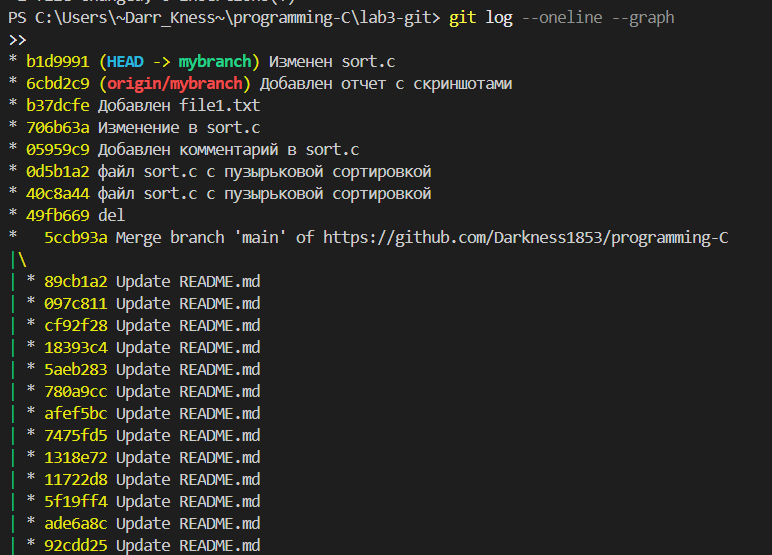
Файл file по-прежнему не отслеживается Git.

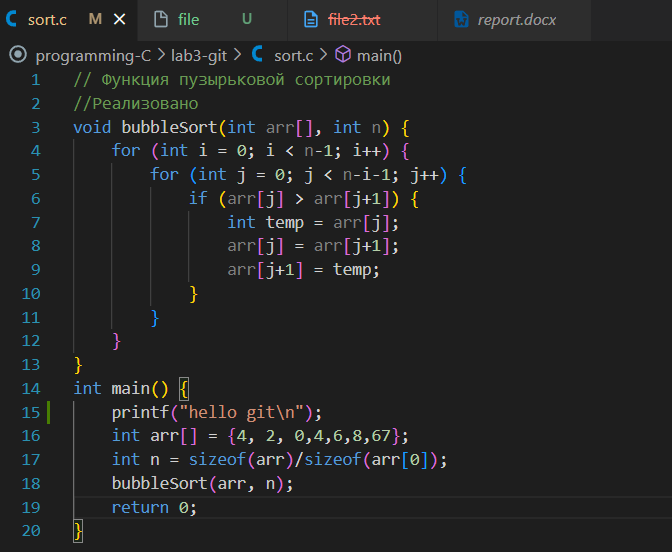
Нет изменений, готовых к коммиту (staging area пуст)

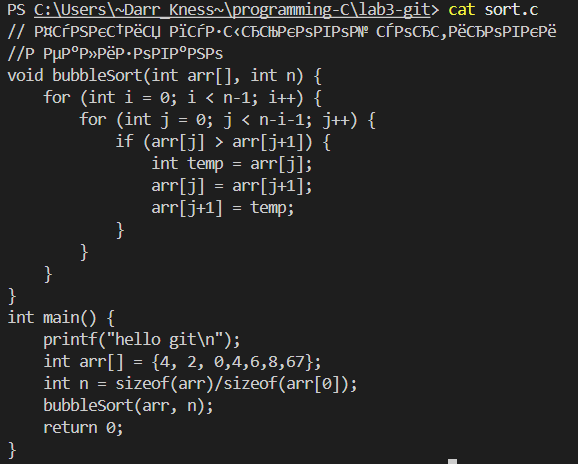
15)Индексируем изменения (add) и делаем коммит

16) Журнал показывает нам, что был последнее изменение было в файле sort.c

и что мы находимся в mybranch.

17)Добавим в sort.c в main() printf(“hello git\n”)

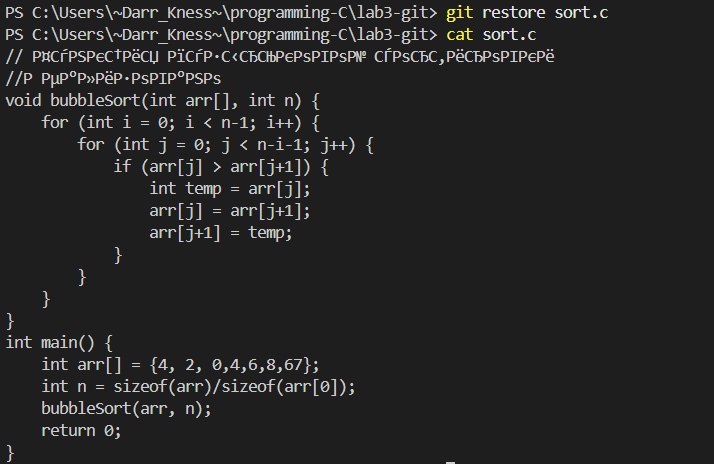
18)Содержимое sort.c



20)Запустим git restore sort.c

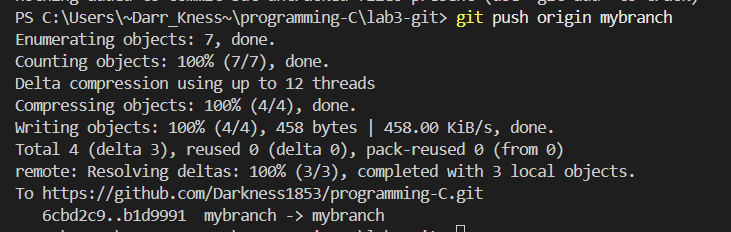
git restore sort.c вернула файл к состоянию последнего коммита, и изменения, которые не были закоммичены, были отменены.

21)Содержимое файла sort.c

22)git status говорит нам, что:

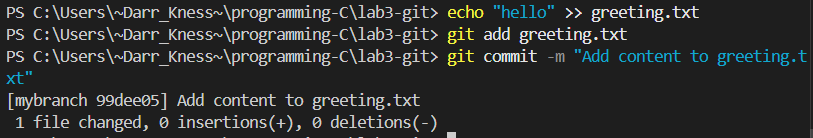
* Мы находитмься на ветке mybranch.
* Моя локальная ветка опережает удалённую ветку origin/mybranch на **1 коммит**.
* В моей рабочей директории есть неотслеживаемый файл file.
* Нет изменений, готовых к коммиту (staging area пуст).

23)Запушитм на удаленный репо ветку с помощью команды git push origin mybranch

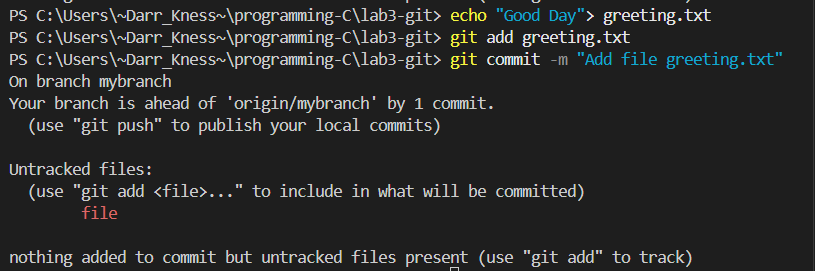


**Ветки:**

1)Создадим файл greeting.txt и закоммитим

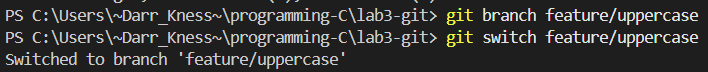


2)Добавим в этот файл слово hello, индексируем и коммитим с текстом "Add content to greeting.txt"

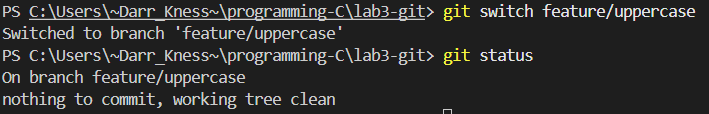


3)Создадим ветку с именем feature/uppercase с помощью команды git branch feature/uppercase

4)Переключимся в нее

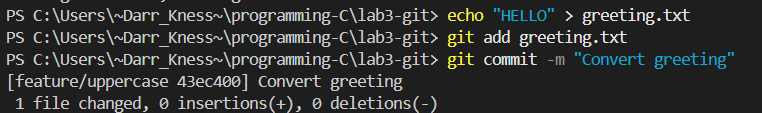


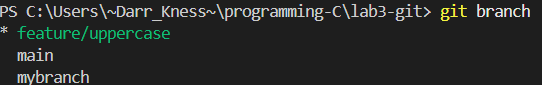
5)Вывод git status



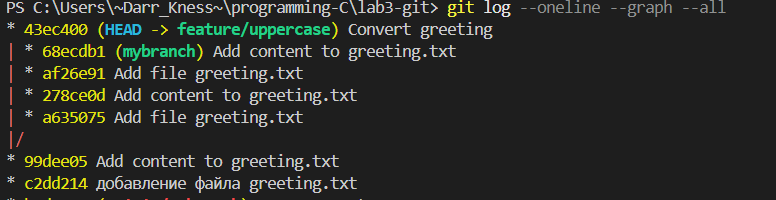
6)Отредактируем greeting.txt, чтобы он содержал приветствие в верхнем регистре (HELLO)

7)Добавим файл greeting.txt и закоммитим

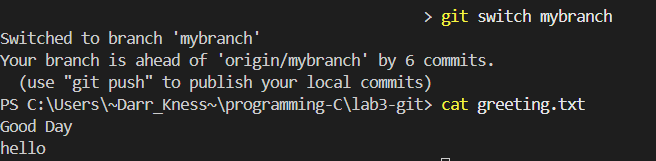
8) git branch выводит информацию:pвёздочка (\*) указывает на текущую ветку(main).

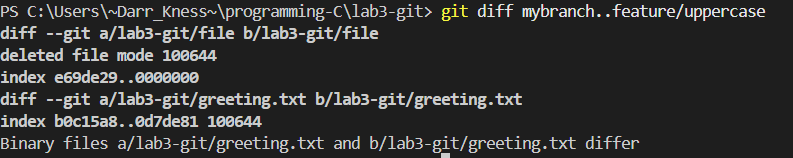
Ветка feature/uppercase существует.

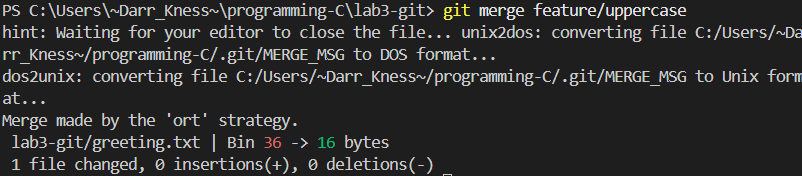
9)Выводим git log --oneline --graph –all

10)Переключимся на главную ветку git switch mybranch

11)Выведем содержимое файла cat greeting.txt

12)Сравним ветки git diff mybranch..feature/uppercase

13)Объединим ветки

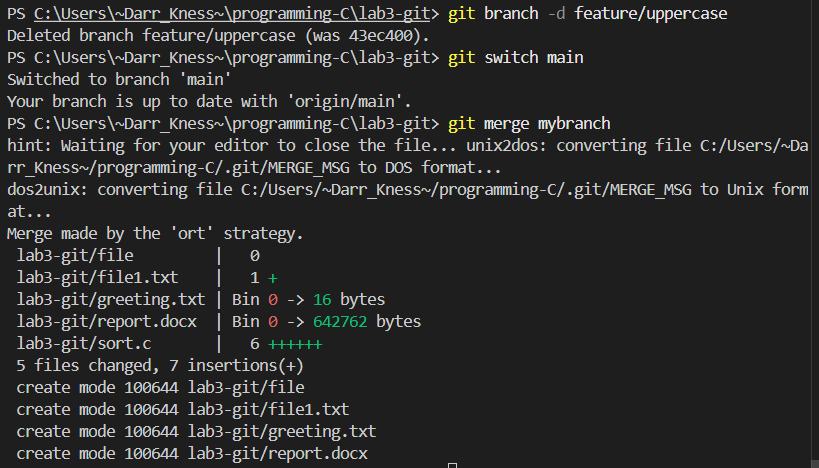
14)Используем cat, чтобы увидеть содержимое файла greetings.txt



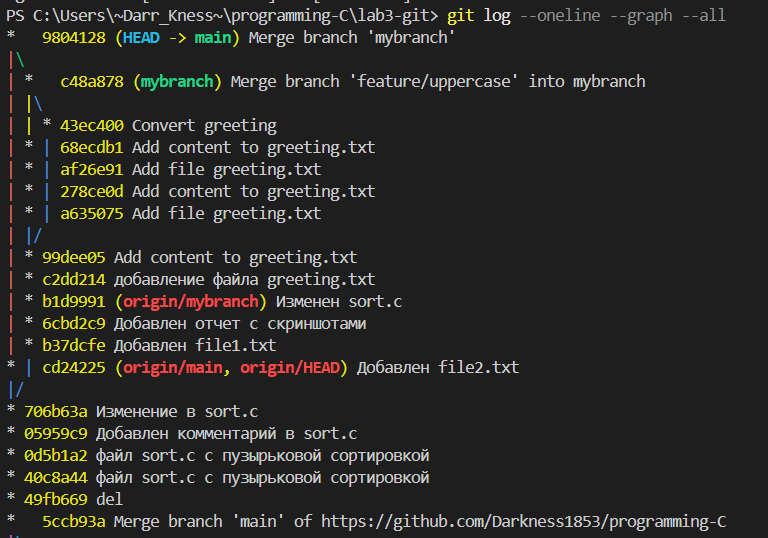
15)Удалим ветку с заглавными буквами (feature/uppercase) с помощью команды

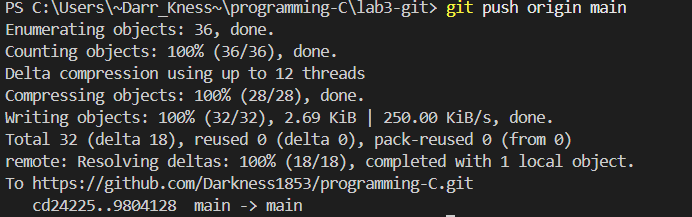
git branch -d feature/uppercase

16)Смержим ветку mybranch в master (git merge)

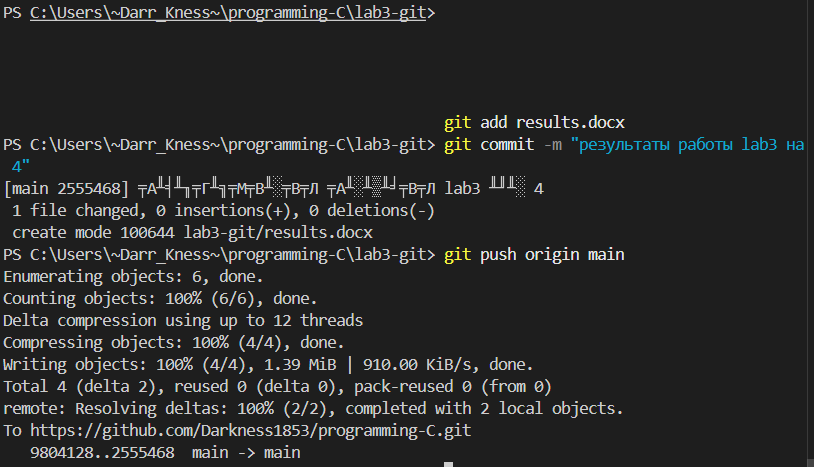


17) git log --oneline --graph --all выводит

18)Запушим изменения ветки master на удаленный репо. git push origin main



19)Запушим документ с результатами нашей работы

**На оценку 5:**

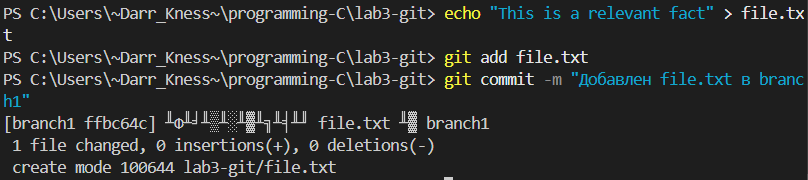
1)Создадим ветку branch1 и переключимся в неё с помощью команд(git branch branch1

git switch branch1)



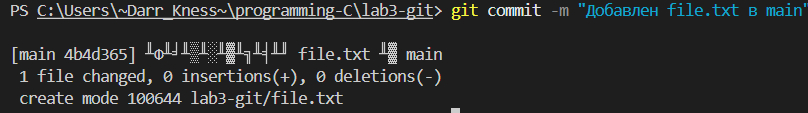
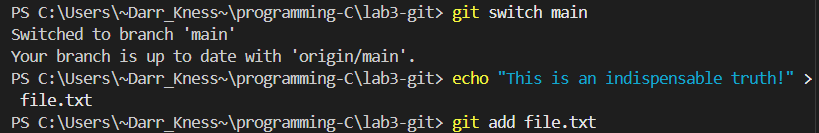
2)Выполним команду $ echo "This is a relevant fact" > file.txt

3)Закоммитим это изменение

4)Переключимся на главную ветку и выполним команду

echo "This is an indispensable truth!" > file.txt

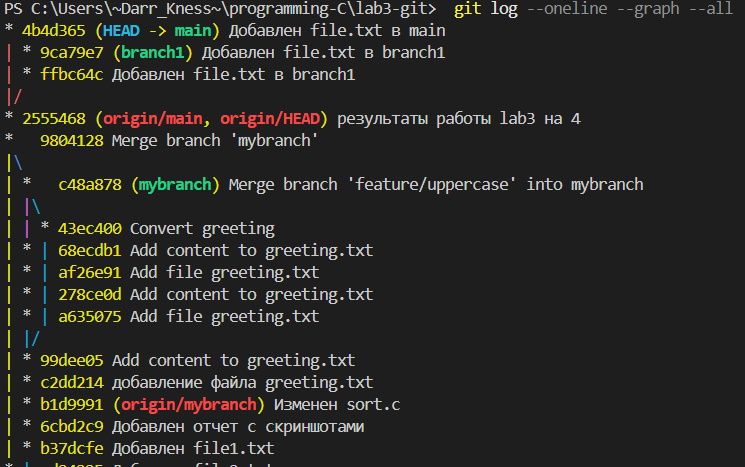
5)Закоммитим изменения в main

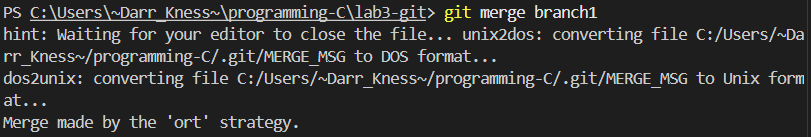


6)Вывод git log --oneline --graph –all

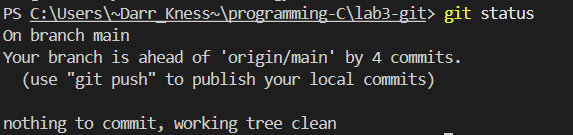
Граф показывает, как ветки ответвляются и сливаются:

* + Ветка branch1 ответвилась от main.
  + Ветка mybranch содержит слияние с веткой feature/uppercase.
  + Ветка main была обновлена после слияния с mybranch.
  + Последний коммит в ветке main, где был добавлен файл file.txt.
  + Последний коммит в ветке branch1, где также был добавлен файл file.txt.
  + Последний коммит в удалённой ветке origin/main, где были зафиксированы результаты работы lab3.
  + Коммит слияния в ветке mybranch, где были объединены изменения из feature/uppercase.

7)Используем команду git merge чтобы смержить ветку branch1 в main



8) git status показывает, что

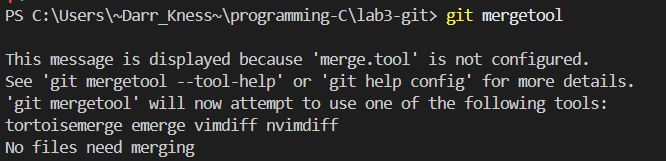


Нечего коммитить , ветвь пуста.

9)Посмотрим содержимое файла file.txt и в любом любимом текстовом

редакторе и исправим конфликт. С помощью git mergetool проверил.

Конфликтов не было обнаружено.

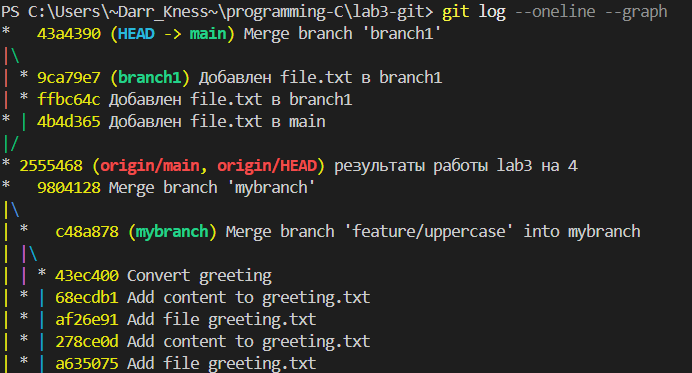
10)git log --oneline –graph показывает, что:

(HEAD -> main): Указывает, что текущая ветка (HEAD) указывает на ветку main.

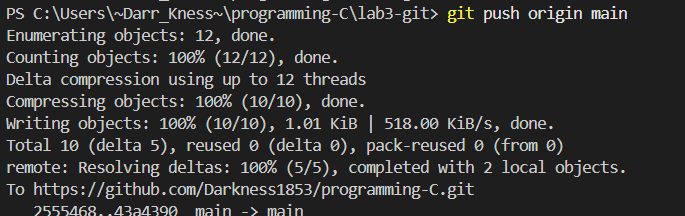
изменения из branch1 были добавлены в main.

Добавлен file.txt в branch1

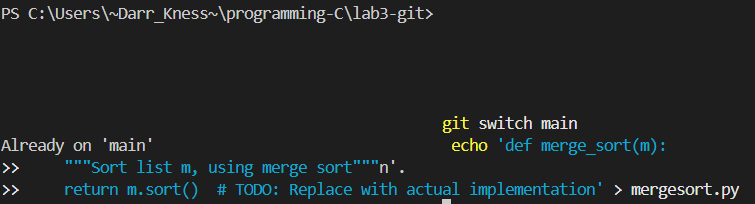
Добавлен file.txt в main

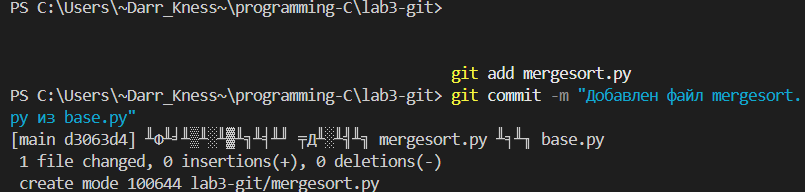


11)Запушим изменения

**Починим Merge конфликты для сортировки MergeSort на python.**

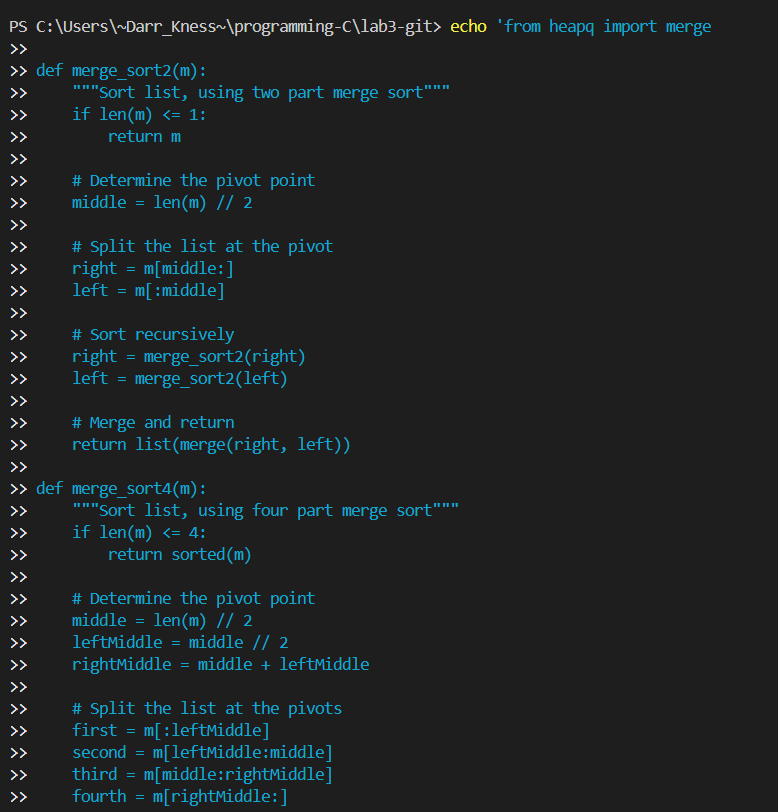
1)Cоздадим файл mergesort.py с содержимым из base.py

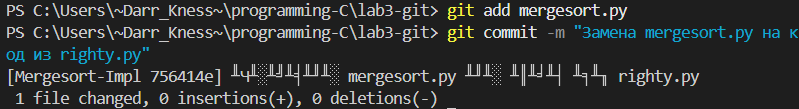
2)Проиндексируем файл и закоммитим

3)Создадим новую сеть и переключится на неё

4)Содержимое файла mergesort.py заменим на код из righty.py

echo 'code' > mergesort.py

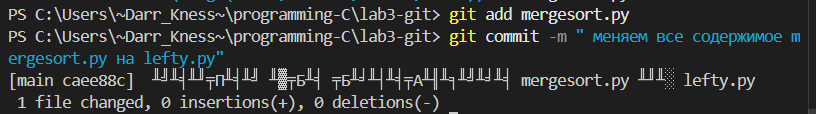
5)Коммитим изменения.



6)Переключаемся на main и меняем все содержимое mergesort.py на lefty.py

echo 'code' > mergesort.py

7)Коммитим изменения.

8) git log --oneline --graph показывает, что:

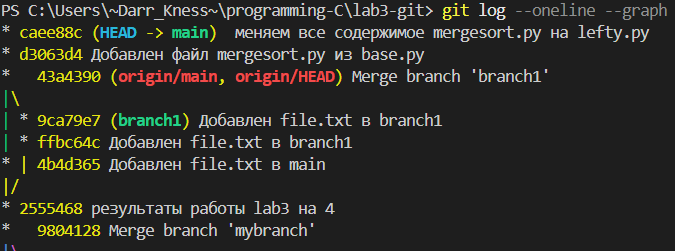
**caee88c (HEAD -> main)**: Это последний коммит в ветке main. В этом коммите содержимое файла mergesort.py было заменено на содержимое файла lefty.py.

**d3063d4**: В этом коммите был добавлен файл mergesort.py, который был взят из base.py

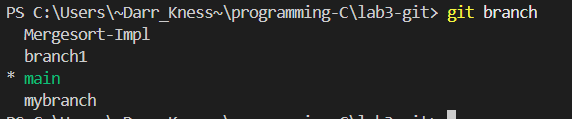
Ветка main идет сверху вниз, и в нее вливаются изменения из других веток (branch1 и mybranch).

Ветка branch1 была активна параллельно с main, и в ней были сделаны два коммита (9ca79e7 и ffbc64c), после чего она была объединена с main.

Ветка mybranch также была объединена с main, и в ней произошло слияние с другой веткой (feature/uppercase).



9) git branch показывает, что

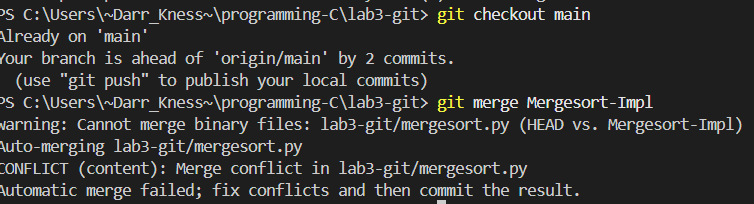


Мы сейчас работаем в ветке main (она активна).

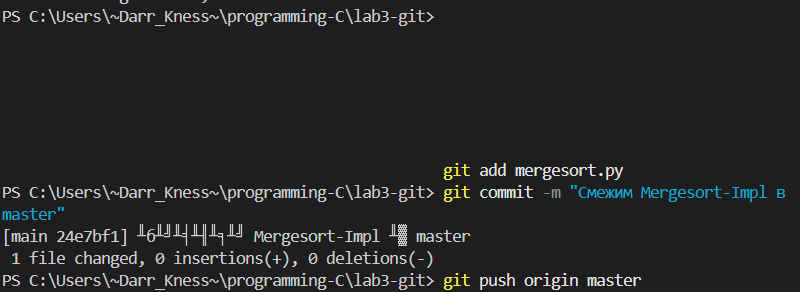
В моем репозитории есть еще три ветки: Mergesort-Impl, branch1 и mybranch.

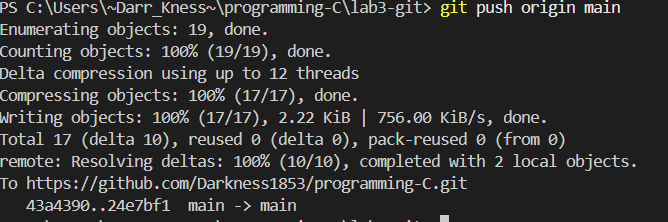
Мы можете переключаться между этими ветками с помощью команды git checkout <имя\_ветки> или git switch <имя\_ветки>.

10)**Необходимо смержить**Mergesort-Impl**в**master

11)Исправим все merge конфликтов запушить в master изменения.

Мы можем обьединить изменения из обеих веток.

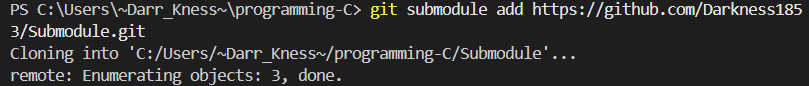
12)Запушим документ с результатами нашей работы



**Создадим сабмодуль**

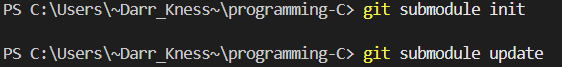
1)Добавим сабмодуль в основной репозиторий с помощью команды

git submodule add https://github.com/Darkness1853/Submodule.git

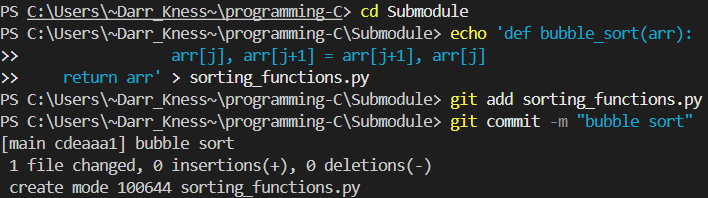
2)Инициализируем сабмодуль с помощью командами:

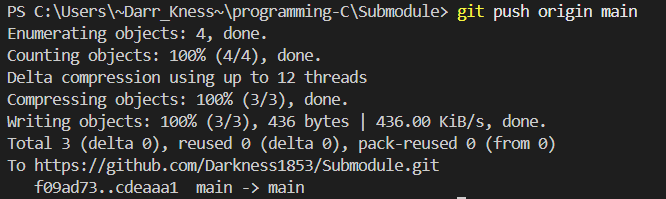
git submodule init

git submodule update



3)Переходим в директорию сабмодуля и добавим файл. Сохраняем и закоммитим

4)Отправляем в главный репозиторий

Конец.

**Памятка(шпаргалка)**

### ****Настройка Git:****

1. git config --global user.name "John Doe"  
   Устанавливает ваше имя для всех Git-репозиториев на компьютере. Это имя будет использоваться в коммитах.
2. git config --global user.email "johndoe@example.com"  
   Устанавливает вашу электронную почту для всех Git-репозиториев. Это нужно для идентификации автора коммитов.
3. git config --global core.editor nano  
   Устанавливает текстовый редактор nano для Git (например, для написания сообщений коммитов).  
   Для Windows можно использовать notepad:  
   git config --global core.editor notepad  
   Это нужно, если вы не хотите использовать vim (который может быть сложным для новичков).

### ****Основные команды:****

1. git add <файл>  
   Добавляет файл в "индекс" (staging area), чтобы Git начал отслеживать изменения в этом файле.
2. git commit  
   Фиксирует изменения, которые вы добавили с помощью git add, и сохраняет их в истории коммитов.  
   Если просто написать git commit, откроется текстовый редактор для написания сообщения коммита.
3. git commit -m "My commit message"  
   Фиксирует изменения с коротким сообщением, описывающим, что было сделано.
4. git log  
   Показывает историю коммитов в текущей ветке.
5. git log -n 5  
   Показывает последние 5 коммитов в истории.
6. git log --oneline  
   Показывает историю коммитов в сокращенном виде (по одной строке на коммит).
7. git log --oneline --graph  
   Показывает историю коммитов в сокращенном виде с графическим отображением веток
8. git log --oneline --graph --all  
   Показывает историю всех коммитов во всех ветках с графическим отображением.

### ****Работа с ветками:****

1. git switch <ветка>  
   Переключает вас на указанную ветку.
2. git switch -c <новая-ветка>  
   Создает новую ветку и сразу переключается на нее.
3. git branch  
   Показывает список всех локальных веток. Текущая ветка будет выделена символом \*.
4. git branch <имя-ветки>  
   Создает новую ветку, но не переключается на нее.
5. git branch -v  
   Показывает список веток с информацией о последнем коммите в каждой.
6. git branch -d <ветка>  
   Удаляет указанную ветку (если она уже слита с другой веткой).
7. git checkout <ветка>  
   Переключает вас на указанную ветку.
8. git checkout -b <новая-ветка>  
   Создает новую ветку и сразу переключается на нее.

### ****Сравнение изменений:****

1. git diff  
   Показывает разницу между текущими изменениями в рабочей директории и последним коммитом.
2. git diff <веткаA> <веткаB>  
   Показывает разницу между двумя ветками.

### ****Отмена изменений:****

1. git restore --staged <файл>  
   Убирает файл из индекса (staging area), но сохраняет изменения в рабочей директории.

### ****Слияние веток:****

1. git merge <ветка>  
   Сливает указанную ветку с текущей.
2. git mergetool --tool=vimdiff  
   Открывает инструмент для разрешения конфликтов слияния (в данном случае vimdiff).

### Команды и их объяснения Submodule

#### 1. ****Команды для работы с подмодулями:****

1. git submodule add <URL> <путь>
2. **Цель**: Добавить подмодуль (внешний репозиторий) в ваш репозиторий по указанному пути.

**Пример**:

ubmodule add https://github.com/Darkness1853/programming-C.git lab3-git/programming-C

**Объяснение**: Добавляет репозиторий programming-C как подмодуль в директорию lab3-git/programming-C.

2.git submodule init

**Цель**: Инициализировать подмодули, указанные в файле .gitmodules.

1. git submodule status
   * **Цель**: Показать состояние всех подмодулей в репозитории.
2. git submodule update
   * **Цель**: Обновить подмодули, загрузив их содержимое.

git submodule update

* + **Объяснение**: Загружает данные подмодулей, если они еще не были загружены.

1. git rm -r <путь>
   * **Цель**: Удалить подмодуль из индекса Git.

git rm -r programming-C

* + **Объяснение**: Удаляет подмодуль programming-C из индекса Git.

1. rm -rf <директория>
   * **Цель**: Удалить локальную копию подмодуля из директории .git/modules.

rm -rf .git/modules/programming-C

* + **Объяснение**: Удаляет подмодуль programming-C с диска.

1. cat .gitmodules
   * **Цель**: Просмотреть содержимое файла .gitmodules.
2. echo '[submodule "lab3-git/programming-C"]' > .gitmodules
   * **Цель**: Создать файл .gitmodules и добавить в него запись для подмодуля.

echo '[submodule "lab3-git/programming-C"]' > .gitmodules

echo ' path = lab3-git/programming-C' >> .gitmodules

echo ' url = https://github.com/Darkness1853/programming-C.git' >> .gitmodules

* + **Объяснение**: Ручное создание или исправление файла .gitmodules.

1. git add .gitmodules
   * **Цель**: Добавить файл .gitmodules в индекс Git.

git add .gitmodules

* + **Ошибка**: fatal: pathspec '.gitmodules' did not match any files.
  + **Причина**: Файл .gitmodules отсутствовал в репозитории.

#### 2. ****Команды для работы с ветками:****

1. git branch
   * **Цель**: Показать список локальных веток.

git branch

* + **Объяснение**: Выводит список всех локальных веток, выделяя текущую ветку символом \*.

1. git checkout <ветка>
   * **Цель**: Переключиться на указанную ветку.

git checkout master

* + **Объяснение**: Переключает рабочую директорию на ветку master.

1. git switch <ветка>
   * **Цель**: Аналог git checkout, но только для переключения веток.

git switch Mergesort-Impl

* + **Объяснение**: Переключает рабочую директорию на ветку Mergesort-Impl.

#### 4. ****Команды для работы с файловой системой:****

1. mkdir <директория>
   * **Цель**: Создать новую директорию.

mkdir lab3-git

* + **Объяснение**: Создает директорию lab3-git.

1. ls
   * **Цель**: Показать содержимое текущей директории.

ls

* + **Объяснение**: Выводит список файлов и папок в текущей директории.

#### 5. ****Команды для работы с удаленными репозиториями:****

1. git clone <URL>
   * **Цель**: Клонировать удаленный репозиторий на локальную машину.

git clone https://github.com/Darkness1853/programming-C.git

* + **Объяснение**: Клонирует репозиторий programming-C в текущую директорию.

#### 6. ****Команды для фиксации изменений:****

1. git add <файл>
   * **Цель**: Добавить файл или директорию в индекс Git.

git add .gitmodules

* + **Объяснение**: Добавляет файл .gitmodules в индекс для последующего коммита.

1. git commit -m "<сообщение>"
   * **Цель**: Зафиксировать изменения в репозитории с указанным сообщением.

git commit -m "Добавлен подмодуль programming-C"

* + **Объяснение**: Создает коммит с сообщением "Добавлен подмодуль programming-C".