|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| *Федеральное государственное бюджетное образовательное учреждение высшего образования* ***«МИРЭА – Российский технологический университет»***  **РТУ МИРЭА** |

Институт Информационных технологий (ИТ)

Кафедра Математического обеспечения и стандартизации информационных технологий (МОСИТ)

|  |
| --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 1** |
| **по дисциплине**  «ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРОГРАММНЫХ ПРИЛОЖЕНИЙ» |
| **Тема: «**Системы контроля версий**»** |

|  |  |
| --- | --- |
| Выполнил студент группы ИКБО-32-21 | Быченков А.К. |
| Принял | Миронов А.И. |

|  |  |  |
| --- | --- | --- |
| Практическая работа выполнена | «\_\_»\_\_\_\_\_\_\_2023 г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_2023 г. | *(подпись руководителя)* |

Москва 2023

**Задание**

**Цель работы:** получить навыки по работе с командной строкой и git’ом. **Часть 1. Основные команды Git**

1. Установите и настройте клиент git на своей рабочей станции.
2. Создайте локальный репозиторий и добавьте в него несколько файлов.
3. Внесите изменения в один из файлов.
4. Проиндексируйте изменения и проверьте состояние.
5. Сделайте коммит того, что было проиндексировано в репозиторий. Добавьте к коммиту комментарий.
6. Измените еще один файл. Добавьте это изменение в индекс git. Измените файл еще раз. Проверьте состояние и произведите коммит проиндексированного изменения. Теперь добавьте второе изменение в индекс, а затем проверьте состояние с помощью команды git status. Сделайте коммит второго изменения.
7. Просмотрите историю коммитов с помощью команды git log. Ознакомьтесь с параметрами команды и используйте некоторые из них для различного формата отображения истории коммитов.
8. Верните рабочий каталог к одному из предыдущих состояний.
9. Изучите, как создавать теги для коммитов для использования в будущем.
10. Отмените некоторые изменения в рабочем каталоге (до и после индексирования).
11. Отмените один из коммитов в локальном репозитории.

**Часть 2. Системы управления репозиториями**

1. Создайте аккаунт на GitHub (у кого нет),
2. Создайте репозиторий на GitHub и на локальной машине, согласно выбранной теме проекта,
3. Создайте несколько файлов на локальной машине при помощи консоли,
4. Создайте SSH-ключ для авторизации,
5. Свяжите репозиторий локальной машины с репозиторием на GitHub при помощи консоли,
6. Создайте новую ветку в репозитории с помощью команды, произведите в ней какие-нибудь изменения, а после слейте с веткой master,
7. Выполните цепочку действий в репозитории, согласно варианту.

|  |  |
| --- | --- |
| № варианта | Задание |
| 4 | 1) Клонируйте непустой удаленный репозиторий на локальную машину  2) Создайте новую ветку и выведите список всех веток  3) Произведите 3 коммита в новой ветке в разные файлы  4) Выгрузите изменения в удаленный репозиторий  5) Откатите в новой ветке предпоследний коммит (в том числе в удаленном репозитории)  6) Выведите в консоли различия между веткой master и новой веткой  7) Слейте новую ветку с master при помощи merge |

**Часть 3. Работа с ветвлением и оформление кода**

1. Сделайте форк репозитория в соответствии с вашим вариантом

|  |  |
| --- | --- |
| № варианта | Репозиторий |
| 4 | <https://github.com/ripienaar/free-for-dev> |

1. Склонируйте его на локальную машину
2. Создайте две ветки branch1 и branch2 от последнего коммита в master'е
3. Проведите по 3 коммита в каждую из веток, которые меняют один и тот же кусочек файла
4. Выполните слияние ветки branch1 в ветку branch2, разрешив конфликты при этом
5. Выгрузите все изменения во всех ветках в удаленный репозиторий
6. Проведите еще 3 коммита в ветку branch1
7. Склонируйте репозиторий еще раз в другую директорию
8. В новом клоне репозитории сделайте 3 коммита в ветку branch1
9. Выгрузите все изменения из нового репозитория в удаленный репозиторий
10. Вернитесь в старый клон с репозиторием, выгрузите изменения с опцией --force
11. Получите все изменения в новом репозитории

**Ход работы**

**Часть 1. Основные команды Git**

На рабочей станции уже был установлен и настроен клиент git. Создадим локальный репозиторий:

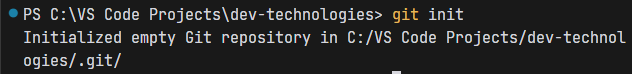


Рисунок 1. Инициализация локального репозитория

Добавим в него несколько файлов:

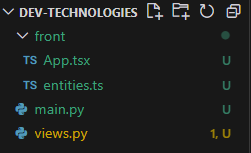


Рисунок 2. Добавленные файлы

В файлы были внесены изменения, с помощью команды git add . проиндексируем эти изменения и проверим состояние командой git status:

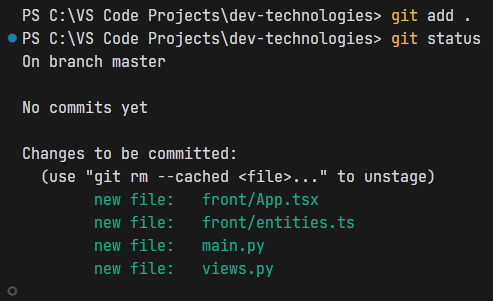


Рисунок 3. Состояние репозитория

Далее сделаем коммит и добавим к нему комментарий:

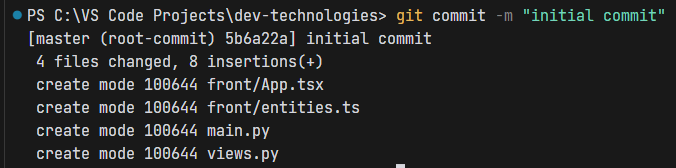


Рисунок 4. Коммит с комментарием

Изменим еще один файл. Добавим это изменение в индекс git. Изменим файл еще раз. Проверим состояние и произведём коммит проиндексированного изменения:

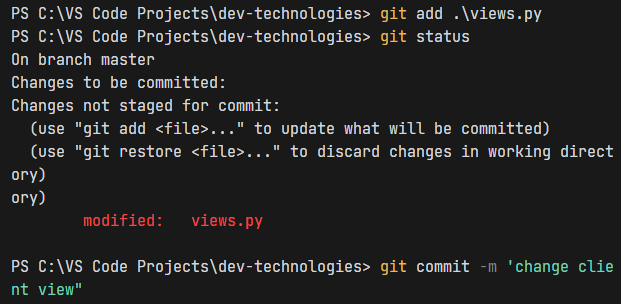


Рисунок 5. Второй коммит

Теперь добавим второе изменение в индекс и проверим состояние с помощью команды git status:

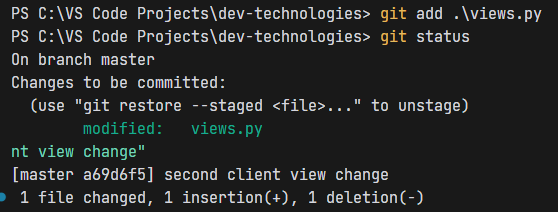


Рисунок 6. Добавление второго изменения в индекс

Сделаем коммит второго изменения и просмотрим историю коммитов с помощью команды git log с аргументом –pretty=oneline для вывода коммита в одну строку:

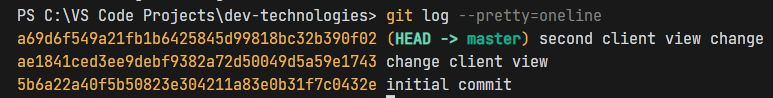


Рисунок 7. История коммитов

Попробуем вернуть рабочий каталог к одному из предыдущих состояний:

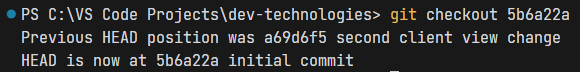


Рисунок 8. Возврат к предыдущему состоянию

Вернёмся к последнему коммиту и создадим для него тег v1:

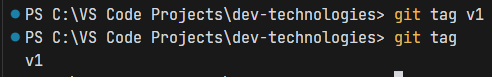


Рисунок 9. Создание тега для коммита

Добавим некоторые изменения в рабочем каталоге и отменим их до индексирования:

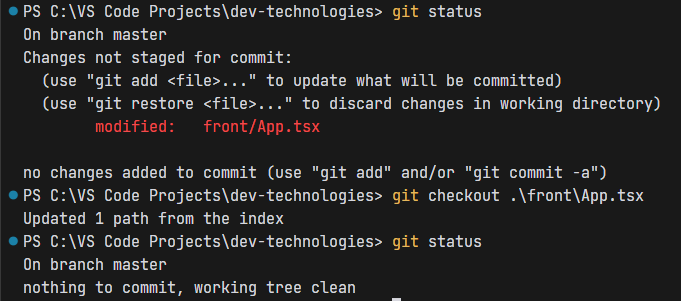


Рисунок 10. Отмена изменений до индексирования

Далее отменим изменения после индексирования:

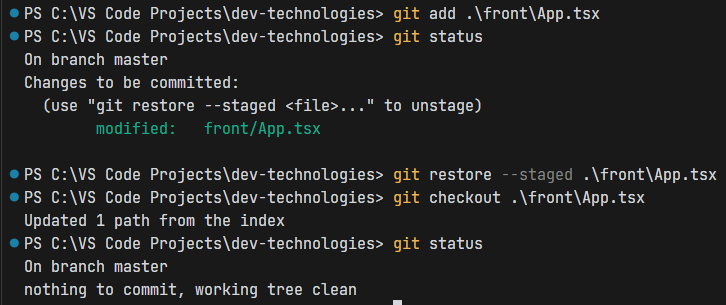


Рисунок 11. Отмена изменений после индексирования

Отменим один из коммитов:

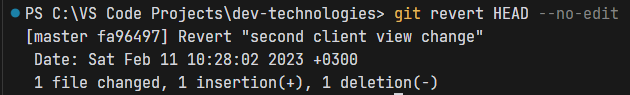


Рисунок 12. Отмена коммита в локальном репозитории

**Часть 2. Системы управления репозиториями**

Аккаунт на GitHub уже был создан и SSH-ключ для авторизации добавлен.

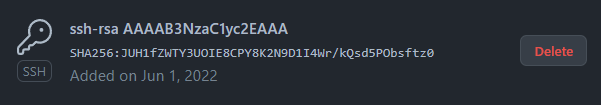


Рисунок 13. SSH-ключ для авторизации

Теперь создадим репозиторий на GitHub:

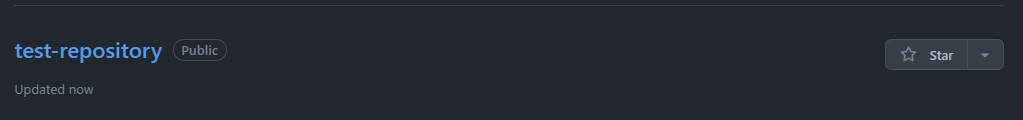


Рисунок 14. Репозиторий на GitHub

Создадим репозиторий на локальной машине и свяжем его с удаленным репозиторием:

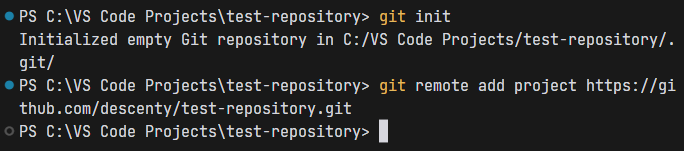


Рисунок 15. Связывание репозиториев при помощи консоли

Создадим несколько файлов на локальной машине при помощи консоли:

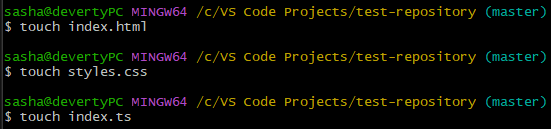


Рисунок 16. Создание нескольких файлов

Создадим новую ветку dev в репозитории с помощью команды.

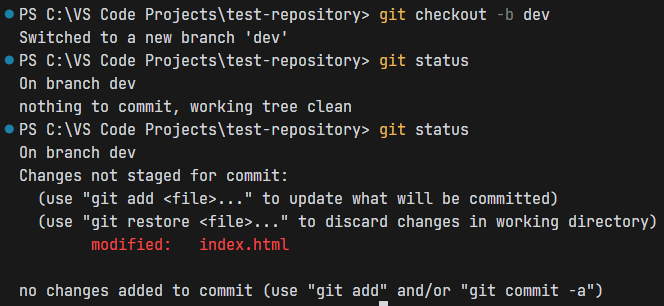


Рисунок 17. Создание новой ветки в репозитории

Произведем в ней некоторые изменения:

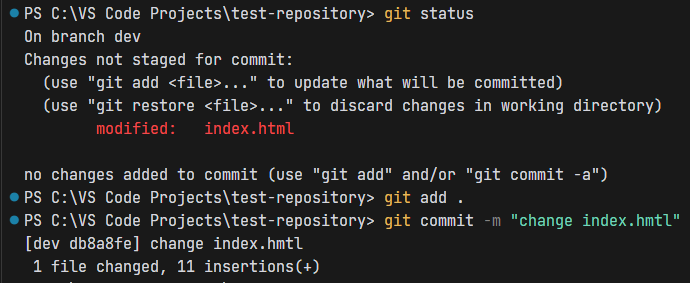


Рисунок 18. Изменения в новой ветке

Теперь сольём ветку dev с веткой main:

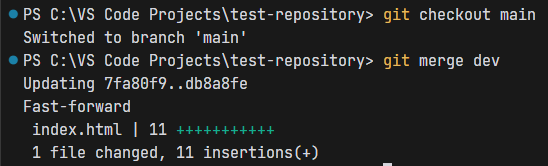


Рисунок 19. Слияние веток

Выгрузим все изменения в удаленный репозиторий:

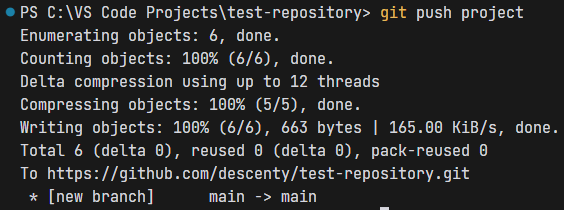


Рисунок 20. Выгрузка изменений в удаленный репозиторий

Далее выполним цепочку действий в репозитории, согласно варианту. Клонируем непустой удаленный репозиторий на локальную машину:

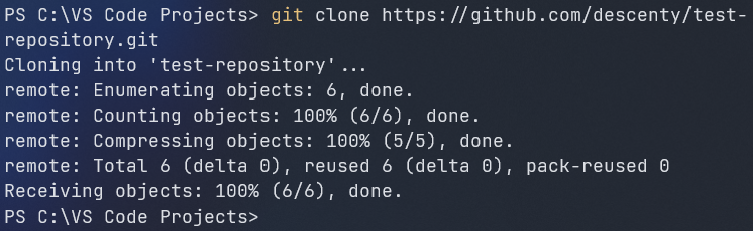


Рисунок 21. Клонирование репозитория

Создадим новую ветку и выведем список всех веток:

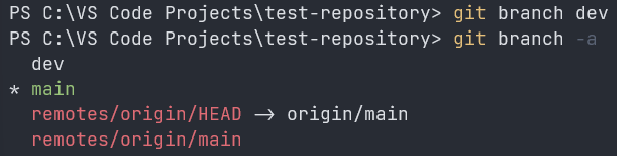


Рисунок 22. Создание новой ветки и вывод всех веток

Произведем 3 коммита в новой ветке в разные файлы:

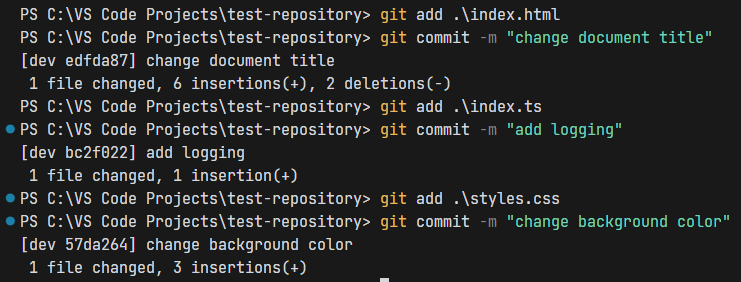


Рисунок 23. Произведенные коммиты в новой ветке

Выгрузим изменения в удаленный репозиторий:

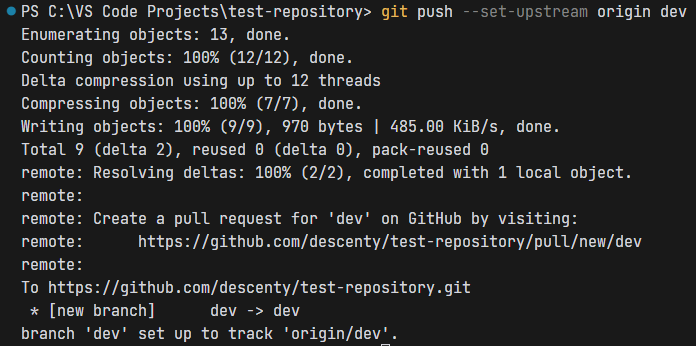


Рисунок 24. Выгрузка изменений в удаленный репозиторий

Откатим в новой ветке предпоследний коммит (в том числе в удаленном репозитории):

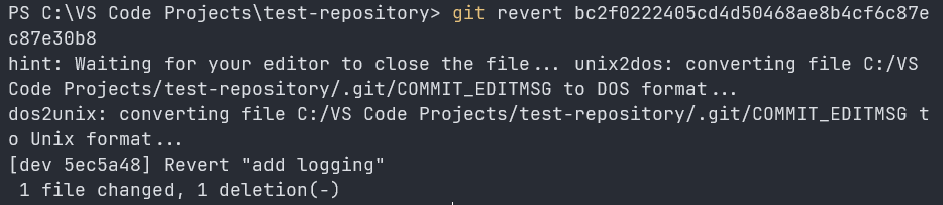


Рисунок 25. Откат препоследнего коммита

Выведем в консоли различия между веткой master и новой веткой:

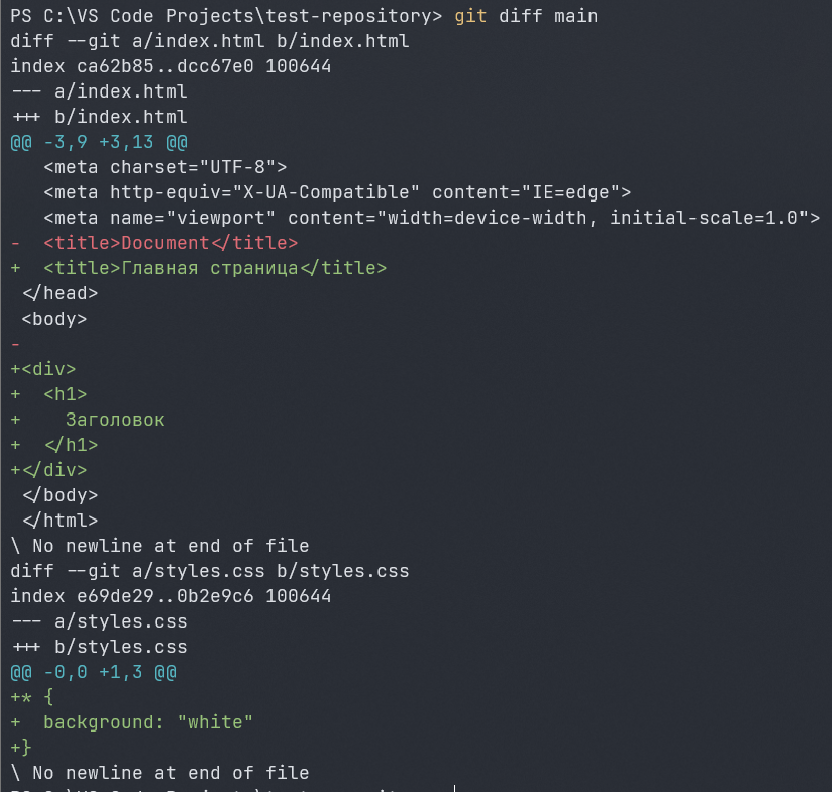


Рисунок 26. Различия между ветками

Сольём новую ветку с master при помощи merge:

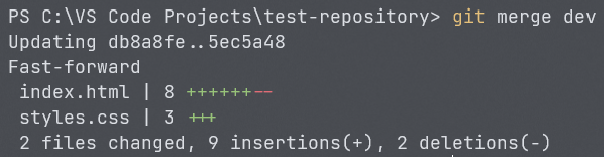


Рисунок 27. Слияние веток

**Часть 3. Работа с ветвлением и оформление кода**

Сделаем форк репозитория согласно варианту:

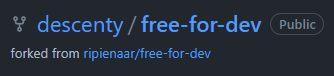


Рисунок 28. Форк репозитория

Склонируем его на локальную машину:

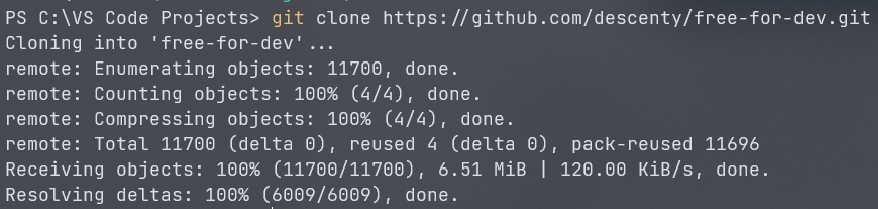


Рисунок 29. Клонирование репозитория

Создадим две ветки branch1 и branch2 от последнего коммита в master’e:



Рисунок 30. Создание веток branch1 и branch2

Проведем по 3 коммита в каждую из веток, которые меняют один и тот же кусочек файла:

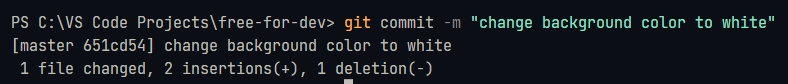


Рисунок 31. Коммит в master

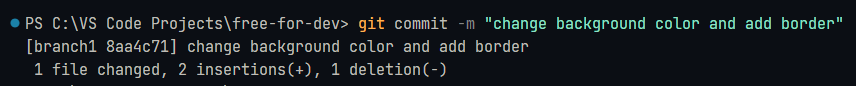


Рисунок 32. Коммит в branch1

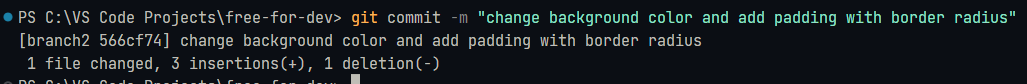


Рисунок 33. Коммит в branch2

Выполним слияние ветки branch1 в ветку branch2 командой git merge. Разрешим конфликты инструментом mergetool:

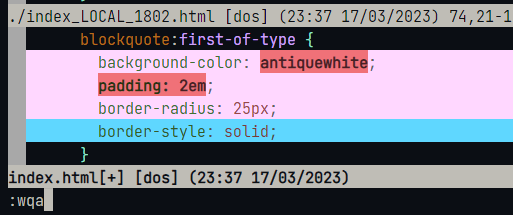


Рисунок 34. Вид mergetool

Рисунок 35. Коммит слияния веток

Выгрузим все изменения во всех ветках в удаленный репозиторий:

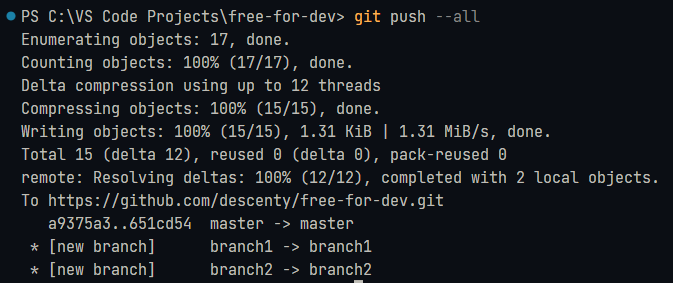


Рисунок 36. Выгрузка всех изменений в удаленный репозиторий

Проведем еще 3 коммита в ветку branch1:

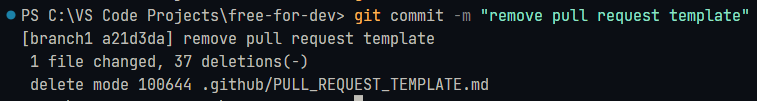


Рисунок 37. Первый коммит

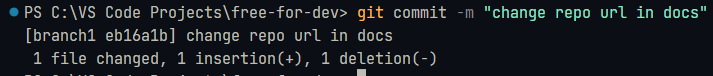


Рисунок 38. Второй коммит

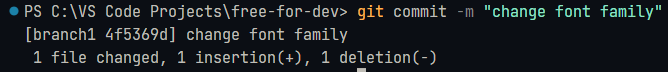


Рисунок 39. Третий коммит

Склонируем репозиторий еще раз в другую директорию:

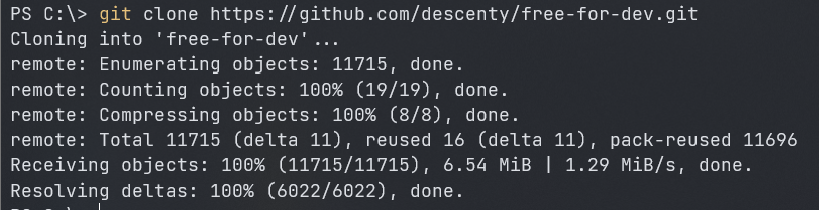


Рисунок 40. Клонирование репозитория

В новом клоне репозитория сделаем 3 коммита в ветку branch1:

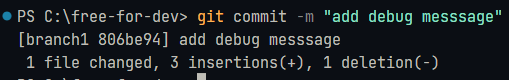


Рисунок 41. Первый коммит

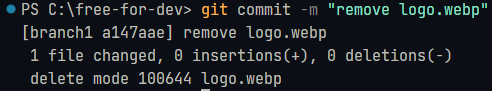


Рисунок 42. Второй коммит

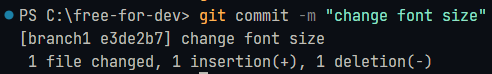


Рисунок 43. Третий коммит

Выгрузим все изменения из нового репозитория в удаленный репозиторий:

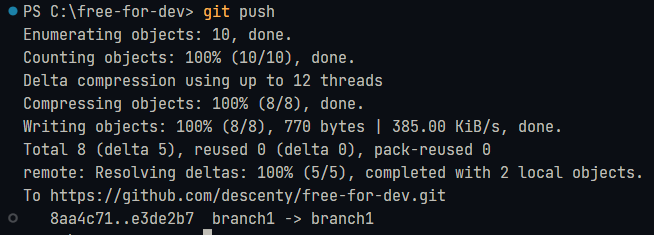


Рисунок 44. Выгрузка изменений

Вернемся в старый клон с репозиторием, выгрузим изменения с опцией –force:



Рисунок 45. Выгрузка изменений с опцией --force

Получим все изменения в новом репозитории:

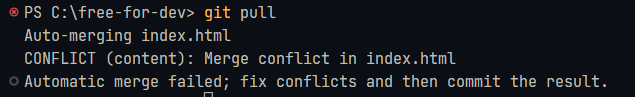


Рисунок 46. Получение изменений

Возник конфликт при слиянии. Разрешим конфликты и создадим новый коммит:

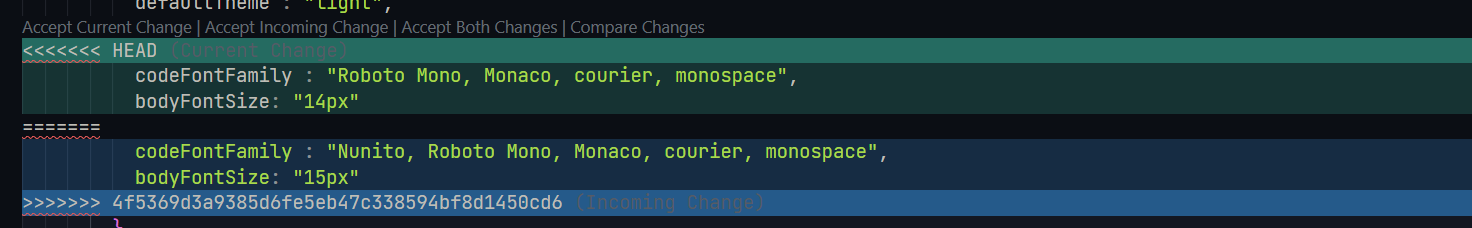


Рисунок 47. Конфликты при слиянии

****

Рисунок 48. Создание коммита слияния

**Ответы на контрольные вопросы**

1. **Что делает команда git diff?**

Команда git diff используется для сравнения изменений между двумя коммитами или между рабочей директорией и индексом (stage), или между рабочей директорией и последним коммитом. Она показывает изменения в виде разницы между двумя состояниями файловой системы.

1. **Что такое pull request?**

Pull request (PR) — это механизм, который позволяет разработчикам предлагать изменения в коде других разработчиков или в основную ветку проекта. При создании PR, разработчик предлагает свои изменения, а другие разработчики могут оставлять комментарии, делать ревью и принимать или отклонять предложенные изменения.

1. **Для чего нужен .gitignore?**

Файл .gitignore используется для указания Git'у, какие файлы и директории должны быть проигнорированы при работе с репозиторием. Например, в .gitignore можно указать, чтобы игнорировались временные файлы, сгенерированные при компиляции кода, файлы с паролями и т.д. Такие файлы могут содержать конфиденциальную информацию и не должны попадать в репозиторий.

1. **Как удалить все untracked файлы?**

Чтобы удалить все untracked файлы, можно использовать команду git clean -f. Она удалит все файлы, которые не были добавлены в индекс, включая новые файлы, временные файлы и т.д.

1. **Чем отличается master и origin/master?**

Ветка master - это основная ветка, которая содержит последнюю стабильную версию кода. В то же время, origin/master — это имя ветки в удаленном репозитории, откуда был склонирован локальный репозиторий.

1. **Как отменить действие команды "git add" на файл?**

Чтобы отменить действие команды git add на файл, можно использовать команду git reset HEAD <file>. Она уберет файл из индекса, но оставит его изменения в рабочей директории. Также можно использовать команду git restore --staged <file>, которая имеет аналогичный эффект.

1. **Как узнать, кто автор строчки в файле, используя систему Git?**

Чтобы узнать, кто автор строчки в файле, используя систему Git, можно использовать команду git blame. Команда git blame показывает для каждой строчки файла, кто был ее последним изменителем, когда и какие изменения были внесены.

1. **Как привести измененный файл в начальное состояние (до изменения)?**

Чтобы привести измененный файл в начальное состояние (до изменения), можно использовать команду git checkout -- <file>. Эта команда откатит изменения файла до последней сохраненной версии в репозитории.

**Вывод**

В результате выполнения работы мною были получены навыки по работе с командной строкой и git’ом.