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»**

|  |  |  |
| --- | --- | --- |
|  | Выполнил:  Омонкулов Исомиддин Валижон угли  2 курс, группа ИВТ-б-о-22-1,  09.03.01 «Информатика и вычислительная техника», направленность (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | |
|  | Руководитель практики:  Воронкин Р. А.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | |
|  | |  | |

Отчет защищен с оценкой \_\_\_\_\_\_\_\_\_\_\_ Дата защиты\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ставрополь, 2024

**Тема**: Основы ветвления Git

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

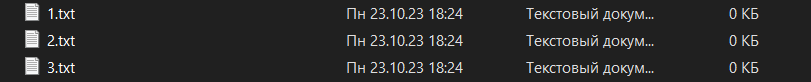


Рисунок 1. Создание трёх текстовых файлов

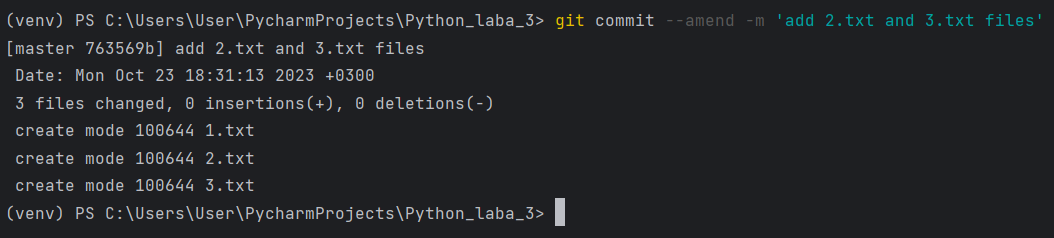


Рисунок 2. Перезапись коммита



Рисунок 3. Создание ветки “my\_first\_branch” и переход на неё

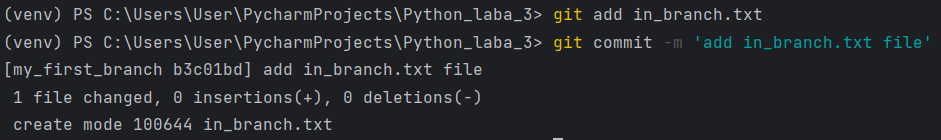


Рисунок 4. Индексация текстового файла “in\_branch” и коммит к нему



Рисунок 5. Переключение на ветку “master”



Рисунок 6. Создание ветки “new\_branch” и переключение на неё

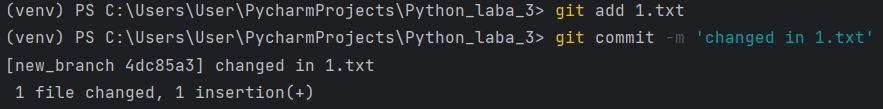


Рисунок 7. Индексация файла “1.txt” и коммит к нему

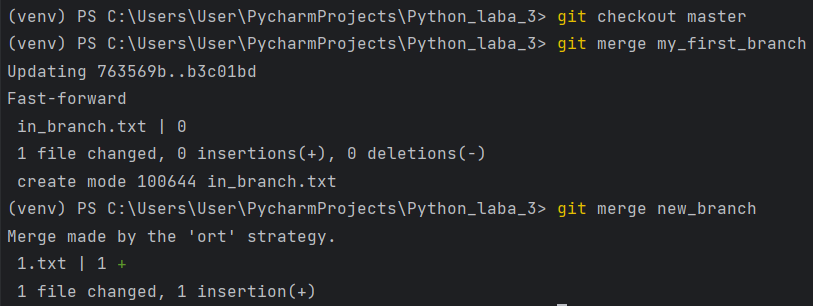


Рисунок 8. Переключение на ветку “master” и слияние с ветками “my\_first\_branch” и “new\_branch”

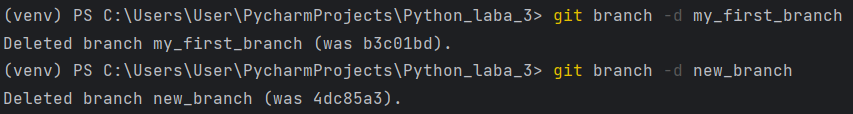


Рисунок 9. Удаление веток “my\_first\_branch” и “new\_branch”



Рисунок 10. Создание веток “branch\_1” и “branch\_2”

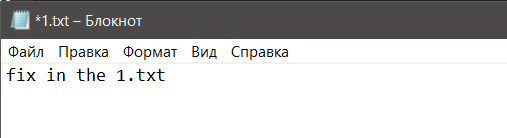


Рисунок 11. Добавил изменение в файл “1.txt”

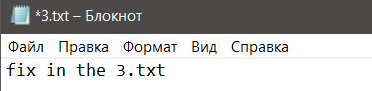


Рисунок 12. Добавил изменения в файл “2.txt”

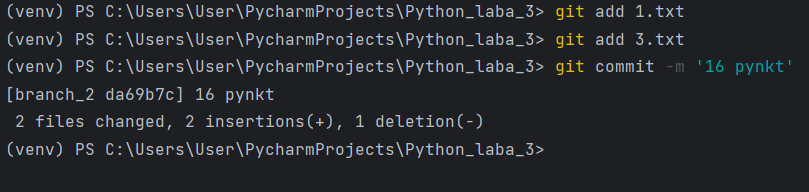


Рисунок 16. Перешёл на ветку “branch\_2”, редактировал файлы “1.txt” и “3.txt”, добавил изменения в индекс и произвёл коммит

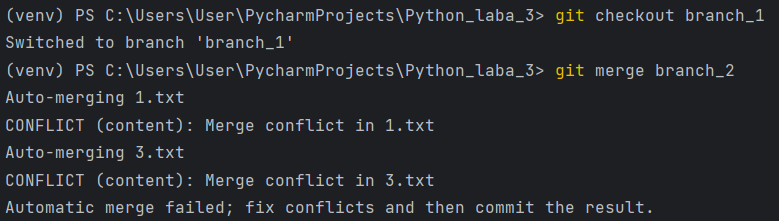


Рисунок 17. Слил изменения ветки “branch\_2” в ветку “branch\_1”, получив два вполне ожидаемых конфликта

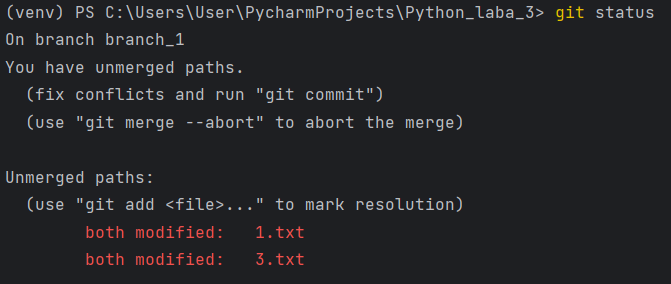


Рисунок 18. Команда “git status”, отображающая конфликты

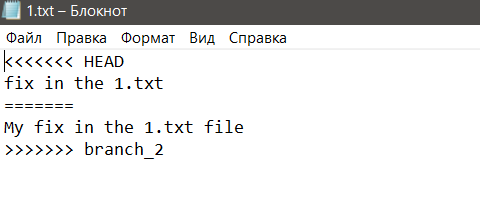


Рисунок 19. Конфликт в файле “1.txt”

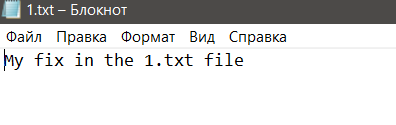


Рисунок 20. Решение первого конфликта вручную

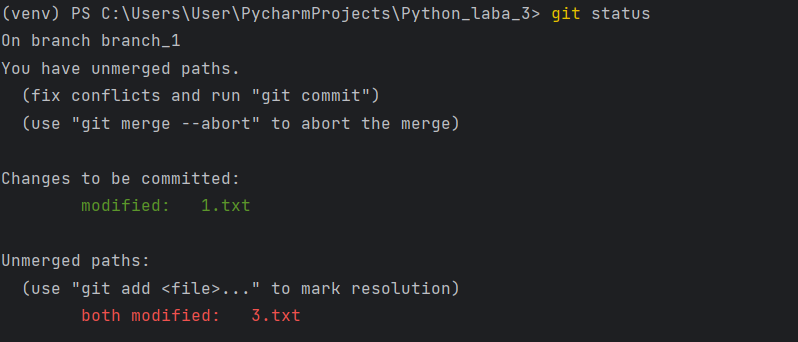


Рисунок 21. Выполнение команды “git status” после решения первого конфликта

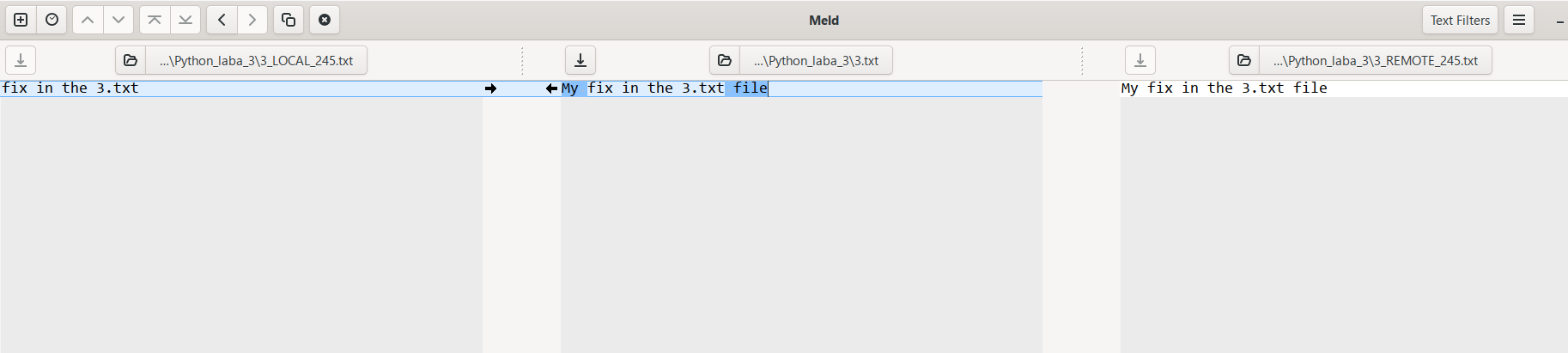


Рисунок 22. Решение второго конфликта с помощью утилиты “Meld”



Рисунок 23. Два успешно урегулированных конфликта



Рисунок 24. Произвёл коммит после решения конфликтов

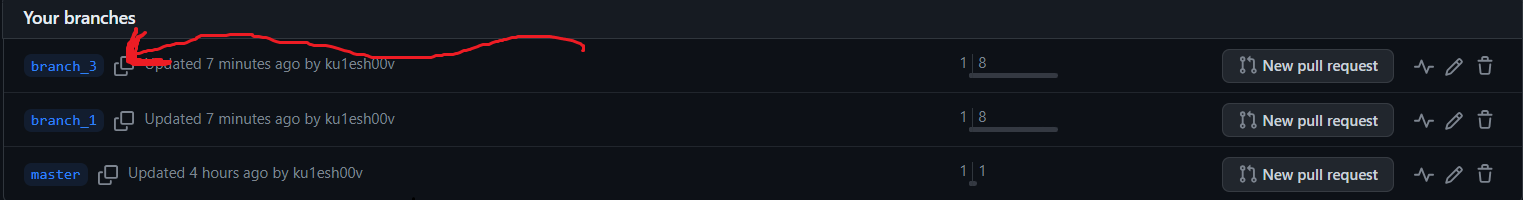


Рисунок 26. Создал ветку “branch\_3” на удалённом репозитории



Рисунок 27. Создал ветку отслеживания удалённой ветки “branch\_3”

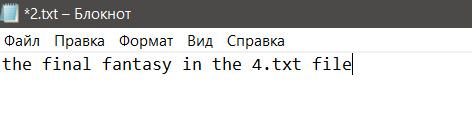


Рисунок 28. После перехода на ветку “branch\_3” внёс изменения в файл “2.txt”

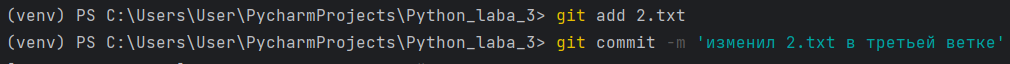


Рисунок 29. Добавил изменения в индекс и произвёл коммит

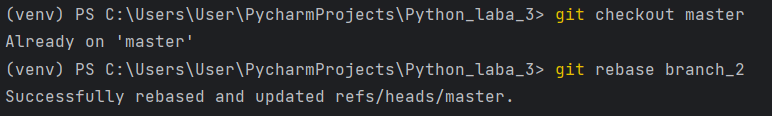


Рисунок 30. Произвёл перебазирование ветки “master” на ветку “branch\_2”

**Вывод:** в ходе выполнения данной лабораторной работы было проведено исследование базовых возможностей при взаимодействии с локальными и удаленными ветками Git.

**Ответы на контрольные вопросы**

1) Ветки в Git представляют собой указатель на коммит. Если нужно добавить какую-то фичу или исправить баг, мы создаём новую ветку. Она будет содержать все изменения, которые мы хотим добавить в репозиторий. После того как работа на функциональностью или исправлением ошибки завершится, эту ветку можно будет объединить с основной веткой репозитория (ещё говорят “влить” или “смержить” ветку).

2)HEAD – это указатель, задача которого ссылаться на определенный коммит в репозитории. Суть данного указателя можно попытаться объяснить с разных сторон.

Во-первых, HEAD – это указатель на коммит в вашем репозитории, который станет родителем следующего коммита.

Во-вторых, HEAD указывает на коммит, относительного которого будет создана рабочая копия во время операции checkout. Другими словами, когда вы переключаетесь с ветки на ветку, используя операцию checkout, то в вашем репозитории указатель HEAD будет переключаться между последними коммитами выбираемых вами ветвей.

3) Чтобы создать новую ветку, необходимо использовать команду git branch branch\_name.

4) Для того, чтобы узнать текущую ветку в гите, можно использовать команду “git branch” в терминале. Текущая ветка будет отмечена звёздочкой (\*). Также можно использовать команду “git status”, которая также покажет текущую ветку.

5) Для переключения на существующую ветку выполните команду git checkout branch\_name

6) Удалённая ветка в Git – это ветка, которая существует на удалённом репозитории, а не на локальной машине разработчика. Это означает, что другие разработчики могут видеть и работать с этой веткой, но она не будет присутствовать в локальном репозитории до тех пор, пока её не будет явно склонировано или получено из удалённого репозитория. Создание удалённой ветки позволяет разработчиками делиться своей работой и совместно работать над проектом.

7) Ветка отслеживания (tracking branch) – в системе контроля версий Git – это ветка, которая связана с удалённой веткой на удалённом репозитории. Она отслеживает изменения в удалённой ветке и автоматически синхронизируется с ней при выполнении операций pull и push.

8) Для создания ветки отслеживания в Git необходимо выполнить команду git checkout с опцией –b и указанием имени локальной ветки и имени удалённой ветки. Например, чтобы создать локальную ветку feature и отслеживать ветку origin/feature, необходимо выполнить команду:

git checkout –b feature origin/feature

9) Чтобы отправить изменения из локальной ветки в удалённую ветку, необходимо выполнить команду git push с указанием имени удалённой ветки. Например, если вы хотите отправить изменения из локальной ветки feature в удалённую ветку origin/feature, выполните следующую команду:

git push origin feature

10) Команда git fetch используется для получения всех изменений из удалённой ветки, но не объединяет их с локальной веткой.

Команда git pull, с другой стороны, получает все изменения из удалённой ветки и автоматически объединяет их с локальной веткой. Таким образом, git pull выполняет как команду git fetch, так и команду объединения (merge) изменений.

Таким образом, основное отличие между git fetch и git pull заключается в том, что git fetch только получает изменения из удалённой ветки, в то время как git pull получает и объединяет их с локальной веткой.

11) Чтобы удалить локальную ветку в Git, используем команду:

Git branch –d branch\_name. Если ветка не была полностью слита с другими ветками, то используйте флаг –D вместо –d для принудительного удаления.

Чтобы удалить удалённую ветку в Git, используйте команду:

Git push origin –delete branch\_name.

Нужно быть осторожным при выполнении данных команд, поскольку эти действия необратимы.

12) В модели git-flow работа с ветками организована следующим образом:

1) Master ветка – используется для хранения стабильной версии приложения.

2. Develop ветка – используется для разработки новых функций и исправлений ошибок.

3. Feature ветки – создаются для разработки конкретных функций. Они отводятся от ветки develop и после завершения работы над функцией сливаются обратно в develop.

4. Release ветки – создаются для подготовки новой версии приложения к релизу. Они отводятся от ветки develop, после завершения подготовки версии сливаются как в master, так и в develop.

5. Hotfix ветки – создаются для исправления критических ошибок в production версии приложения. Они отводятся от ветки master и после исправления сливаются как в master, так и в develop.

Недостатки git-flow включают в себя:

1. Сложность – модель got-flow может быть слишком сложной для небольших команд или проектов.

2. Долгий цикл релизов – из-за использования отельных release и hotfix веток, цикл релизов может затягиваться.

3. Ненужная сложность – для некоторых проектов модель git-flow может быть излишне сложной и накладывать дополнительные ограничения на команду разработчиков.

12) Одним из программных средств с GUI для работы с git является GitKraken. В GitKraken можно легко создавать, переключаться между ветками, удалять и сливать их. Инструмент предоставляет удобный визуальный интерфейс для работы с ветками, что делает процесс управления ветками более интуитивным и удобным для пользователей. Кроме того, GitKraken предоставляет возможность просмотра истории коммитов, создания pull reqest’ов и многое другое, что делает работу с ветками в git более эффективной.