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 ПРАКТИЧЕСКОЙ РАБОТЕ №3**

**дисциплины «Программирование на Python»**

**Вариант**

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Выполнил:  Говоров Егор Юрьевич  2 курс, группа ИВТ-б-о-22-1,  09.03.02 «Информатика и вычислительная техника», направленность (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | |
|  | | Руководитель практики:  Воронкин.Р.А, канд. технических наук, доцент кафедры инфокоммуникаций  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (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. Создание 3 txt файлов

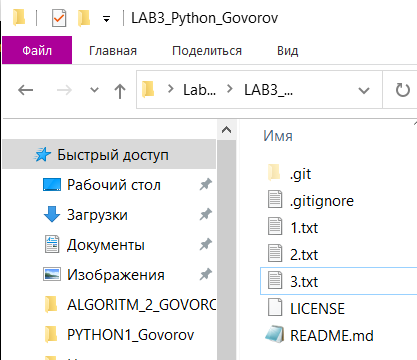


Рис 1. Созданные файлы

1. Проиндексировал первый файл и сделал коммит

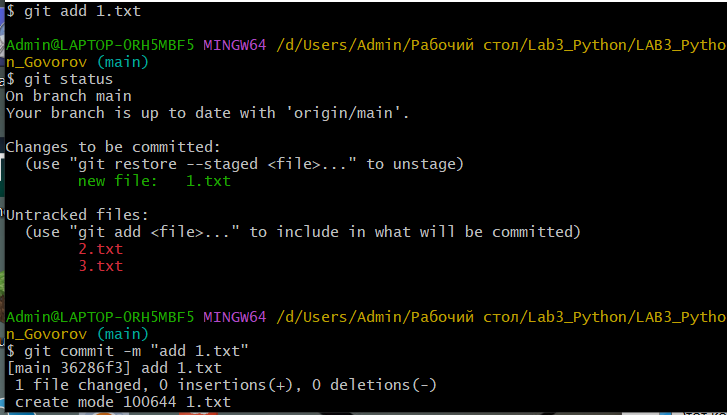


Рис 2. Индексация первого файла и его коммит

1. Проиндексировал второй и третий файл и сделал коммит

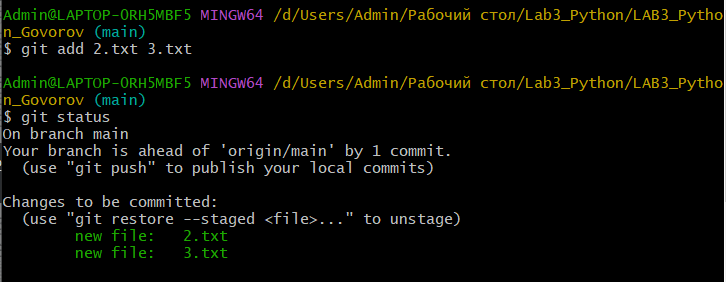


Рис 3. Индексация второго и третьего файла и их коммит

1. Перезаписал уже сделанный коммит с новыми файлами

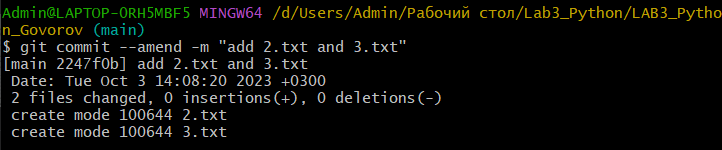


Рис 4. Перезаписанный файл

1. Создал новую ветку

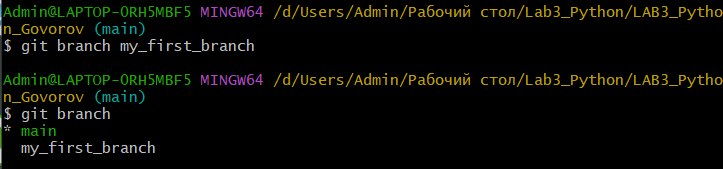


Рис 5. Создание новой ветки

1. Перешёл на ветку и создал новый файл, закоммитил изменения.

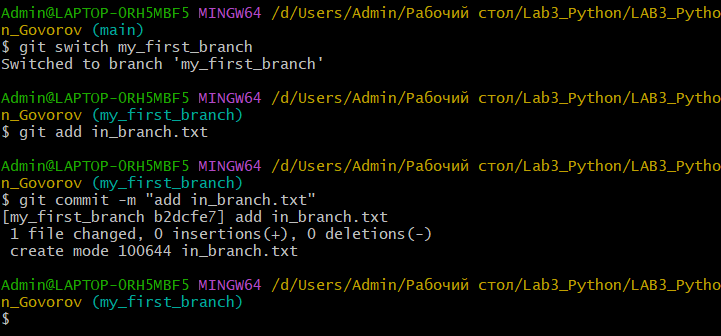


Рис 6. Переход и создание файла на новой ветке

1. Вернулся на основную ветку, создал новую и зашел на нее

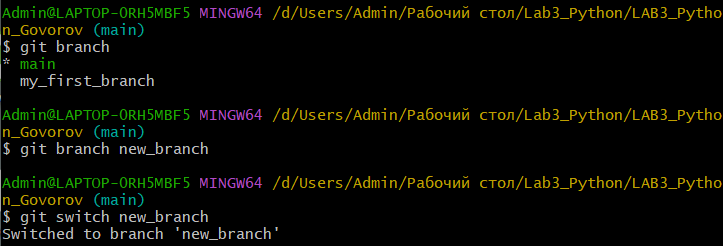


Рис 7. Переход на new\_branch

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

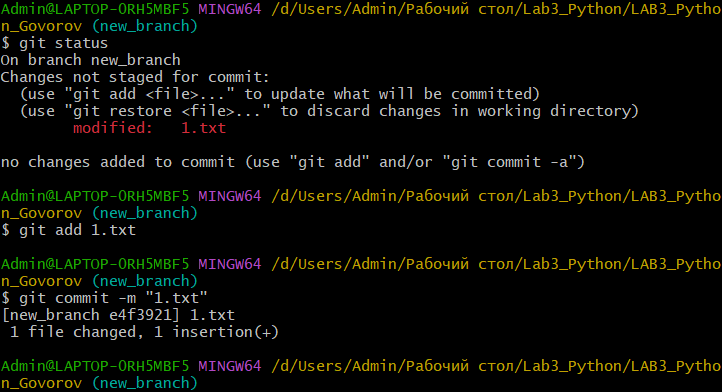


Рис 8. Изменения в 1.txt

1. Перешёл на ветку main и слил ветки main и my\_first\_branch, после чего слил ветки main и new\_branch.

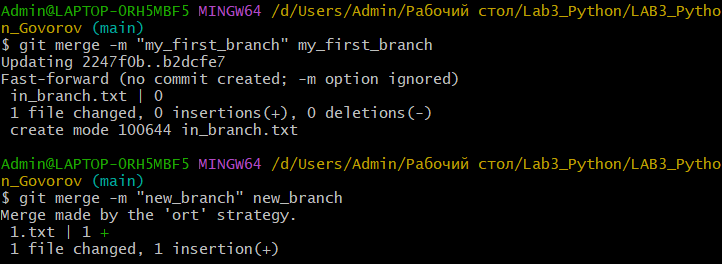


Рис 9. Слил ветки в одну

1. Удалил созданные ветки

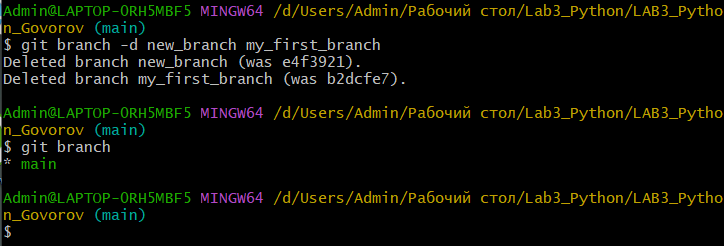


Рис 10. Удаление веток

1. Перешёл на ветку branch\_1 и изменил файл 1.txt, удалив все содержимое и добавив текст “fix in the 1.txt”. Сделал измения файла 3.txt, удалив все содержимое и добавив текст “fix in the 3.txt”, закоммитил изменения

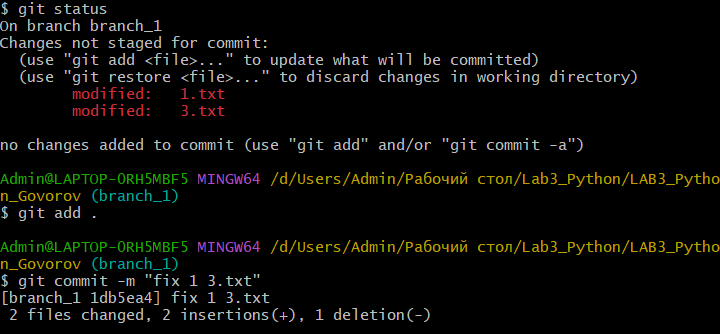


Рис 11. Изменения в 1.txt и 2.txt

1. Слил изменения ветки branch\_2 в ветку branch\_1

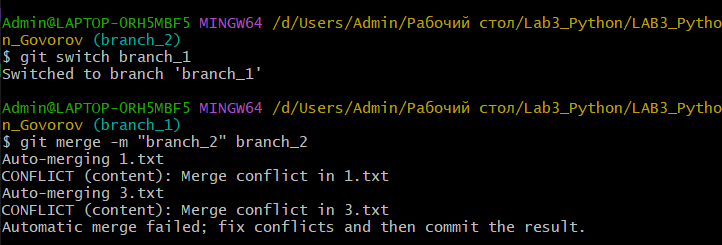


Рис 12. Слияние веток

1. Решил конфликт двух файлов

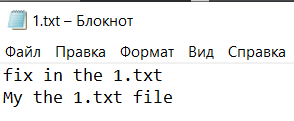


Рис 13. Решение вручную

1. Отправил ветку branch\_1 на Github

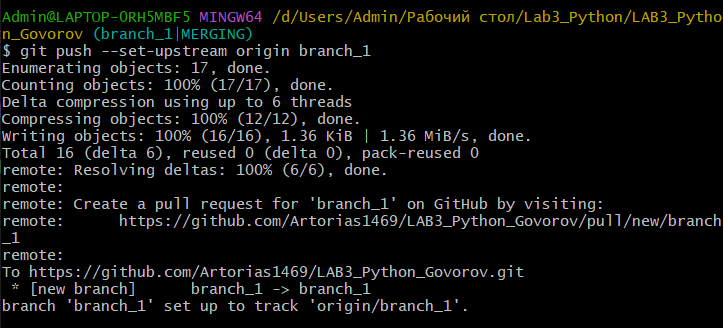


Рис 14. Отправление ветки на Github

1. Создал удаленную ветку branch\_3

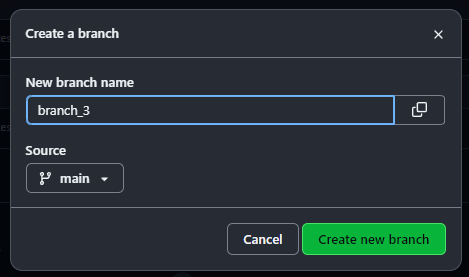


Рис 15. Создал удаленную ветку

1. Создал в лок. реп отслеживание удаленной ветки branch\_3

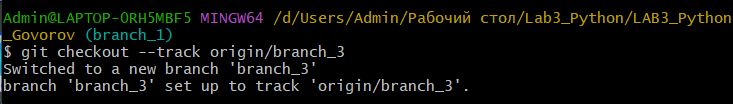


Рис 16. Создал отслеживаемую ветку

1. На ветке branch\_3 в документ 2.txt добавил "the final fantasy in the 4.txt file"

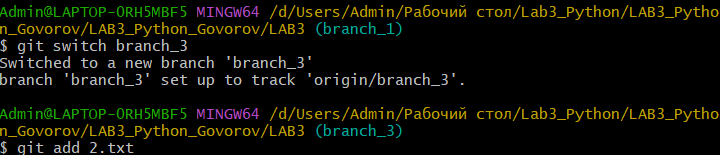


Рис 17. Изменения в 2.txt

1. Переместил ветку master на branch\_2

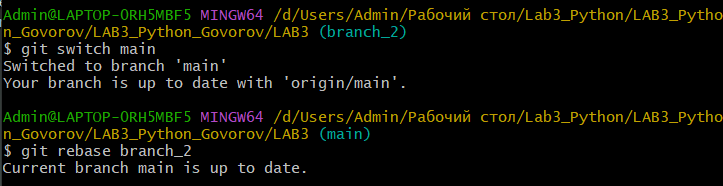


Рис 18. Перемещение ветки

Ответы на контрольные вопросы:

1. Что такое ветка? Под веткой принято понимать независимую последовательность коммитов в хронологическом порядке. Однако конкретно в Git реализация ветки выполнена как указатель на последний коммит в рассматриваемой ветке. После создания ветки уже новый указатель ссылается на текущий коммит.

2. Что такое HEAD? HEAD в Git-это указатель на текущую ссылку ветви, которая, в свою очередь, является указателем на последний сделанный вами коммит или последний коммит, который был извлечен из вашего рабочего каталога. HEAD – 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 . 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. Способы создания веток.

1) Команда git branch: Создание новой ветки без переключения на нее;

2) команда git checkout -b: Создание и переключение на новую ветку одной командой;

3) создание веток в удаленных репозиториях (GitHub): веб-интерфейс позволяет создавать ветки и отправлять их в удаленный репозиторий.

4. Как узнать текущую ветку?

С помощью команд git branc и git status.

5. Как переключаться между ветками?

С помощью команд git checkout, git switch и git branch

6. Что такое удаленная ветка?

Удаленная ветка - это ветка, которая существует в удаленном репозитории и отслеживает состояние истории изменений в этом удаленном репозитории. Она может быть доступна для скачивания и обновления изменений между вашим локальным репозиторием и удаленным репозиторием. Удаленные ветки используются для совместной работы и синхронизации изменений между разными разработчиками и репозиториями.

7. Что такое ветка отслеживания?

Ветки слежения — это ссылки на определённое состояние удалённых веток. Это локальные ветки, которые нельзя перемещать; Git перемещает их автоматически при любой коммуникации с удаленным репозиторием, чтобы гарантировать точное соответствие с ним. Ветка отслеживания - это локальная ветка в Git, которая непосредственно связана с удаленной веткой. Ветка отслеживания автоматически отслеживает изменения в удаленной ветке и позволяет синхронизировать локальные изменения с удаленным репозиторием.

8. Как создать ветку отслеживания?

Для создания ветки отслеживания в Git, вы можете использовать команды git checkout или git switch с флагом -t (или --track).

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

git push remote\_name local\_branch\_name:remote\_branch\_name remote\_name: Имя удаленного репозитория, куда вы хотите отправить изменения (обычно это "origin"). local\_branch\_name: Имя вашей локальной ветки, из которой вы отправляете изменения. remote\_branch\_name: Имя удаленной ветки, в которую вы хотите отправить изменения.

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

Команда git fetch загружает все изменения из удаленного репозитория в ваш локальный репозиторий, но не автоматически объединяет их с вашей текущей веткой. Это означает, что git fetch не изменяет вашу рабочую директорию и не создает новых коммитов в текущей ветке. Вместо этого он обновляет информацию о состоянии удаленных веток, которая хранится локально. После выполнения git fetch, вы можете решить, какие изменения объединить (если это необходимо) и когда. Команда git pull также загружает изменения из удаленного репозитория в ваш локальный репозиторий, но, в отличие от git fetch, она автоматически пытается объединить эти изменения с вашей текущей веткой. git pull фактически объединяет изменения из удаленной ветки в вашу текущую ветку и создает новый коммит, если это необходимо. Это может привести к конфликтам слияния, если ваша текущая ветка и удаленная ветка имеют конфликтующие изменения.

11. Как удалить локальную и удаленную ветки?

Для удаления локальной ветки используется команда git branch -d с именем ветки, которую вы хотите удалить. Удаление веток на удалённом сервере выполняется при помощи команды git push origin --delete

12. Изучить модель ветвления git-flow (использовать материалы статей https://www.atlassian.com/ru/git/tutorials/comparing-workflows/gitflowworkflow, https://habr.com/ru/post/106912/). Какие основные типы веток присуствуют в модели git-flow? Как организована работа светками в модели git-flow? В чем недостатки git-flow? Модель git-flow предполагает следующие основные типы веток:

1. Main (Master) Branch\*\*: Главная ветка, в которой хранится стабильная и готовая к продакшн версия продукта.

2. Develop Branch\*\*: Ветка разработки, в которой объединяются новые функции и исправления из разных веток фичей. Здесь происходит основная разработка.

3. Feature Branches\*\*: Ветки фичей, создаются для разработки новых функций. Каждая фича имеет свою собственную ветку, которая создается от ветки `develop` и после завершения фичи сливается обратно в `develop`.

4. Release Branches\*\*: Ветки релизов, создаются перед выпуском новой версии. В них можно проводить финальное тестирование и подготовку к релизу. После завершения релиза ветка сливается как в `develop`, так и в `main` (для обновления стабильной версии).

5. Hotfix Branches\*\*: Ветки исправлений, создаются для быстрого исправления критических ошибок в текущей стабильной версии (ветке `main`). После исправления ошибки ветка сливается как в `develop`, так и в `main`. Работа с ветками в модели git-flow организована так: - Фичи создаются от `develop`. - Релизные ветки создаются перед выпуском новой версии и сливаются как в `main`, так и в `develop` после завершения тестирования. - Хотфиксы создаются от `main` для исправления критических ошибок и сливаются как в `main`, так и в `develop` после исправления.

Недостатки git-flow: 1. Сложность: Модель git-flow может быть слишком сложной для небольших проектов или команд, где требуется более простой подход к управлению ветками.

2. Замедление разработки: Создание множества дополнительных веток (фичей, релизов, хотфиксов) может замедлить процесс разработки и увеличить сложность слияния изменений.

3. Ветвление релизов: Ветки релизов могут стать сложными и требовать много усилий при долгосрочной разработке, особенно если между ними происходит много изменений.

4. Стандарт не всегда подходит: Модель git-flow не всегда идеально подходит для всех видов проектов и может потребовать адаптации к конкретным потребностям.

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. Создание веток: GitHub Desktop позволяет создавать новые локальные ветки на основе существующих веток в вашем репозитории. Вы можете указать имя и базовую ветку для новой ветки. Переключение между ветками: Вы можете легко переключаться между локальными ветками с помощью интерфейса GitHub Desktop. Текущая активная ветка отображается в верхней части приложения. Отслеживание удаленных веток: GitHub Desktop отображает доступные удаленные ветки для вашего репозитория. Вы можете создавать локальные отслеживающие ветки для удаленных веток и синхронизировать изменения. Просмотр истории веток: Инструмент предоставляет визуальное отображение истории изменений в ваших ветках. Вы можете просматривать коммиты и их связи между ветками. Слияние веток: GitHub Desktop поддерживает слияние изменений из одной ветки в другую. Вы можете выполнить слияние локальных веток или изменений из удаленных веток. Удаление веток:Вы можете удалять локальные ветки с помощью GitHub Desktop. Также есть возможность удаления удаленных веток (после подтверждения).

Вывод: В процессе выполнения лабораторной работы были проведены исследования базовых возможностей по работе с локальными и удаленными ветками Git.