

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 №1.3
дисциплины «Программирование на Python»

Выполнил:
Степанов Леонид Викторович
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение
средств вычислительной
техники и автоматизирование
систем», очная форма обучения

(подпись)

Руководитель практики:
Воронкин Р.А., канд. техн. наук,
доцент, доцент кафедры
инфокоммуникаций

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2023 г.

Тема: основы ветвления GIT

Цель: исследование базовых возможностей по работе с локальными и удалёнными ветками Git


Порядок выполнения работы:

2. Создаём новый репозиторий

Create a new repository



A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (*).

Owner *  FiaLDI / Repository name *
 lab1.3 is available.

Great repository names are short and memorable. Need inspiration? How about [potential-lamp](#) ?

Description (optional)

- ☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.
- ☐  **Private**
You choose who can see and commit to this repository.

Initialize this repository with:

- ☒ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

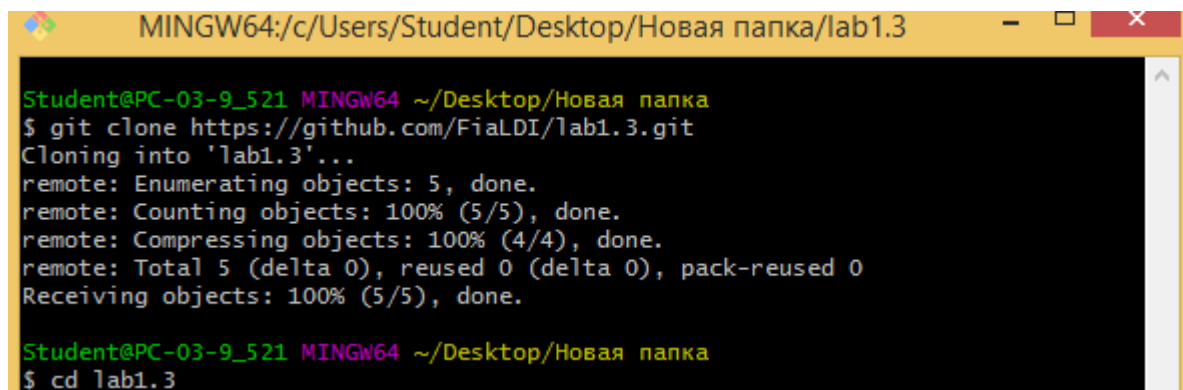
A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

This will set  **main** as the default branch. Change the default name in your [settings](#).

 You are creating a public repository in your personal account.

Create repository

Рисунок 1 - Создание репозитория



```
MINGW64:/c/Users/Student/Desktop/Новая папка/lab1.3
Student@PC-03-9_521 MINGW64 ~/Desktop/Новая папка
$ git clone https://github.com/FiaLDI/lab1.3.git
Cloning into 'lab1.3'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.
Student@PC-03-9_521 MINGW64 ~/Desktop/Новая папка
$ cd lab1.3
```

Рисунок 2 - Клонирование репозитория

3. Создаём три текстовых файла: 1.txt, 2.txt, 3.txt

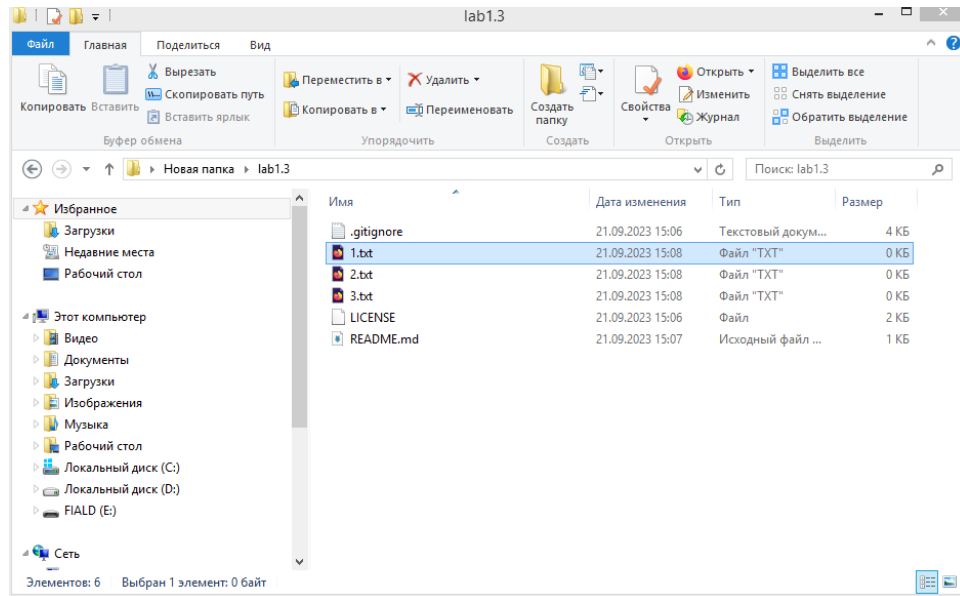


Рисунок 3 – Локальный репозиторий

4. Индексируем 1.txt и делаем коммит

```
MINGW64:/c/Users/Student/Desktop/Новая папка/lab1.3

Student@PC-03-9_521 MINGW64 ~/Desktop/Новая папка
$ git clone https://github.com/FiaLDI/lab1.3.git
Cloning into 'lab1.3'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.

Student@PC-03-9_521 MINGW64 ~/Desktop/Новая папка
$ cd lab1.3

Student@PC-03-9_521 MINGW64 ~/Desktop/Новая папка/lab1.3 (main)
$ git add 1.txt

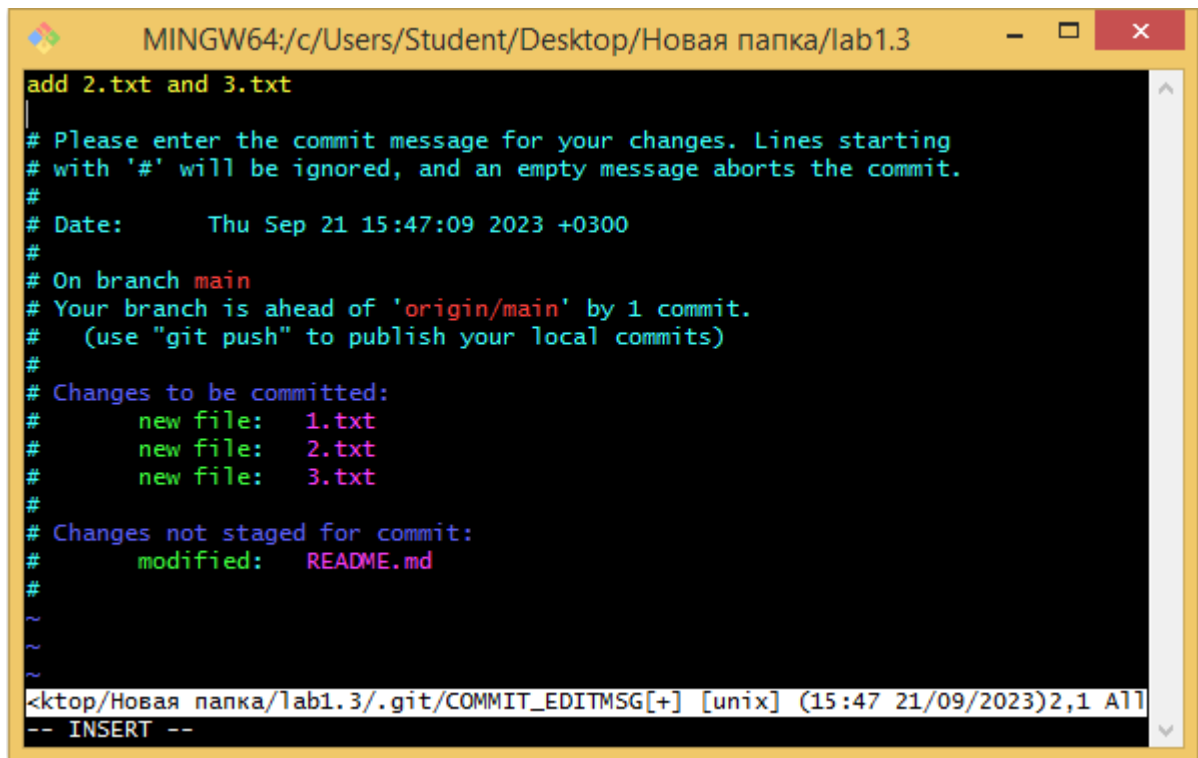
Student@PC-03-9_521 MINGW64 ~/Desktop/Новая папка/lab1.3 (main)
$ git commit -m "add 1.txt file"
[main 805acb8] add 1.txt file
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 1.txt

Student@PC-03-9_521 MINGW64 ~/Desktop/Новая папка/lab1.3 (main)
$
```

Рисунок 4 – Создание коммита

5. Индексируем 2.txt и 3.txt

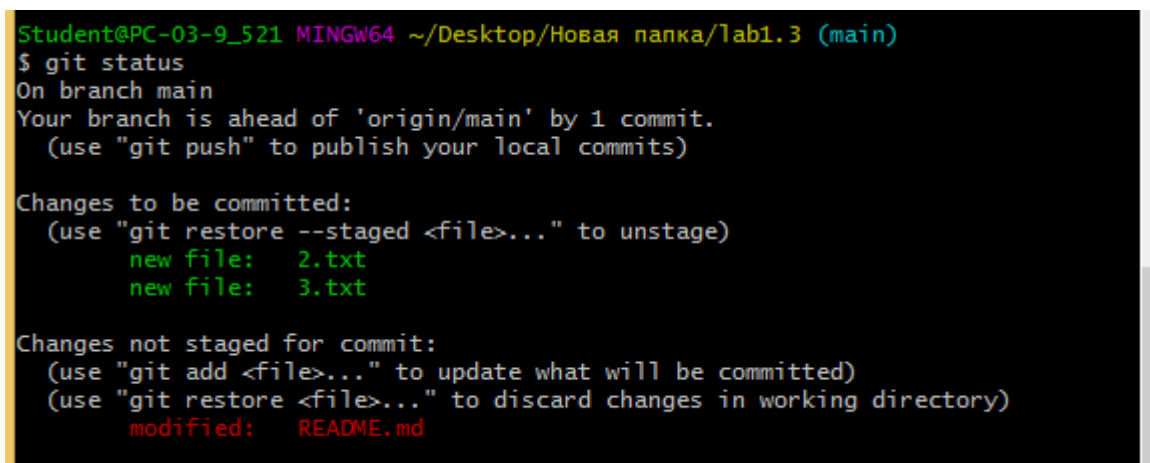
6. Изменяем коммит



```
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:      Thu Sep 21 15:47:09 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
#
# Changes not staged for commit:
#   modified:   README.md
#
~
~
~
<ktop/Новая папка/lab1.3/.git/COMMIT_EDITMSG[+] [unix] (15:47 21/09/2023)2,1 All
-- INSERT --
```

Рисунок 5 – Изменение коммита



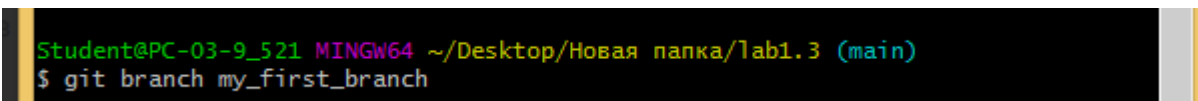
```
Student@PC-03-9_521 MINGW64 ~/Desktop/Новая папка/lab1.3 (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   2.txt
    new file:   3.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   README.md
```

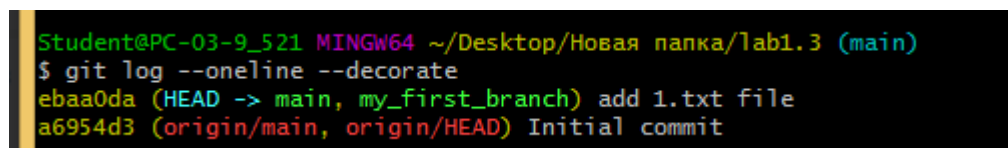
Рисунок 6 – Проверка изменений

7. Создал новую ветку



```
Student@PC-03-9_521 MINGW64 ~/Desktop/Новая папка/lab1.3 (main)
$ git branch my_first_branch
```

Рисунок 7 – Создание новой ветки



```
Student@PC-03-9_521 MINGW64 ~/Desktop/Новая папка/lab1.3 (main)
$ git log --oneline --decorate
ebaa0da (HEAD -> main, my_first_branch) add 1.txt file
a6954d3 (origin/main, origin/HEAD) Initial commit
```

Рисунок 8 – Проверка создания новой ветки

8. Переходим на новую ветку и создаём файл (in_branch.txt), который КОММИТИМ

```
Student@PC-03-9_521 MINGW64 ~/Desktop/Новая папка/lab1.3 (main)
$ git checkout my_first_branch
Switched to branch 'my_first_branch'
A       2.txt
A       3.txt
M       README.md
```

Рисунок 9 - Переход на новую ветку

```
Student@PC-03-9_521 MINGW64 ~/Desktop/Новая папка/lab1.3 (my_first_branch)
$ git commit -m "add in_branch.txt"
[my_first_branch b308f7f] add in_branch.txt
3 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 2.txt
create mode 100644 3.txt
create mode 100644 in_branch.txt
```

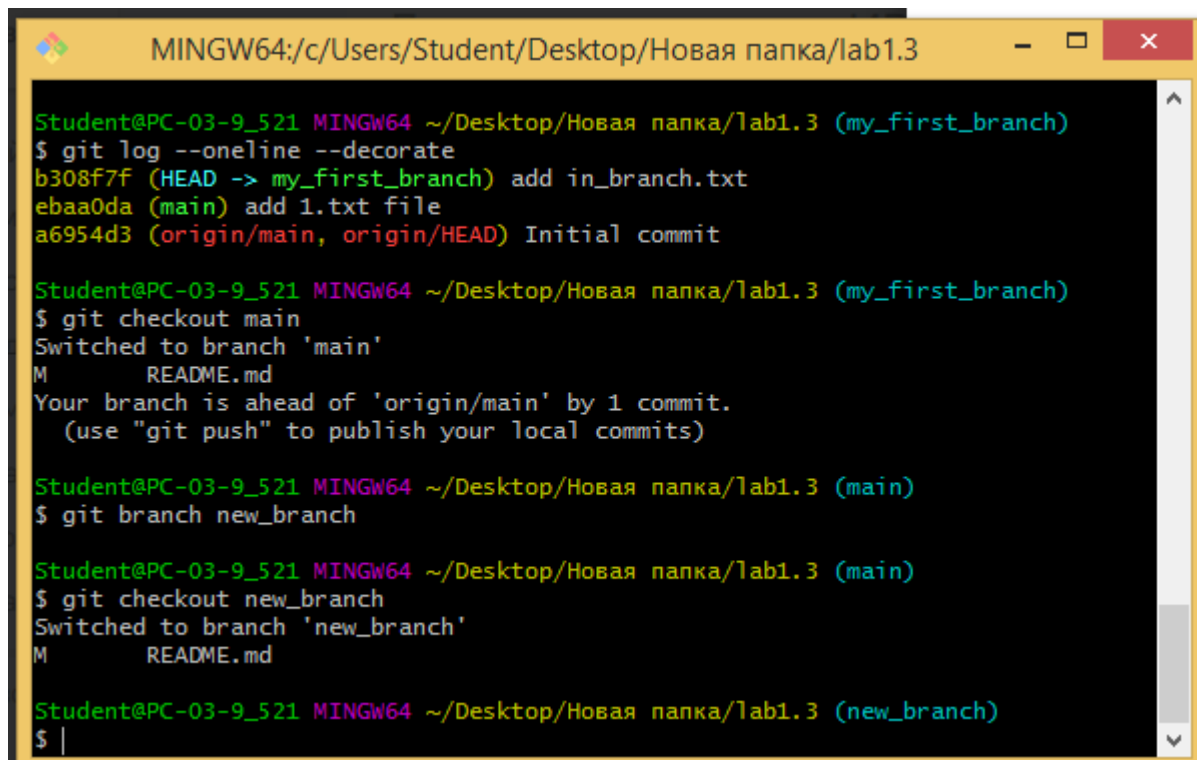
Рисунок 10 - Создание коммита в ветке my_first_branch

9. Переходим на ветку main

```
Student@PC-03-9_521 MINGW64 ~/Desktop/Новая папка/lab1.3 (my_first_branch)
$ git checkout main
Switched to branch 'main'
M       README.md
Your branch is ahead of 'origin/main' by 1 commit.
(use "git push" to publish your local commits)
```

Рисунок 11 - Переход на ветку main

10. Создаём новую ветку (new_branch) и переходим на неё



```
Student@PC-03-9_521 MINGW64 ~/Desktop/Новая папка/lab1.3 (my_first_branch)
$ git log --oneline --decorate
b308f7f (HEAD -> my_first_branch) add in_branch.txt
ebaa0da (main) add 1.txt file
a6954d3 (origin/main, origin/HEAD) Initial commit

Student@PC-03-9_521 MINGW64 ~/Desktop/Новая папка/lab1.3 (my_first_branch)
$ git checkout main
Switched to branch 'main'
M       README.md
Your branch is ahead of 'origin/main' by 1 commit.
(use "git push" to publish your local commits)

Student@PC-03-9_521 MINGW64 ~/Desktop/Новая папка/lab1.3 (main)
$ git branch new_branch

Student@PC-03-9_521 MINGW64 ~/Desktop/Новая папка/lab1.3 (main)
$ git checkout new_branch
Switched to branch 'new_branch'
M       README.md

Student@PC-03-9_521 MINGW64 ~/Desktop/Новая папка/lab1.3 (new_branch)
$
```

Рисунок 12 – Создание и переход на репозиторий new_branch

11. Делаем изменения в файле (1.txt) добавляем текст и коммитим изменения

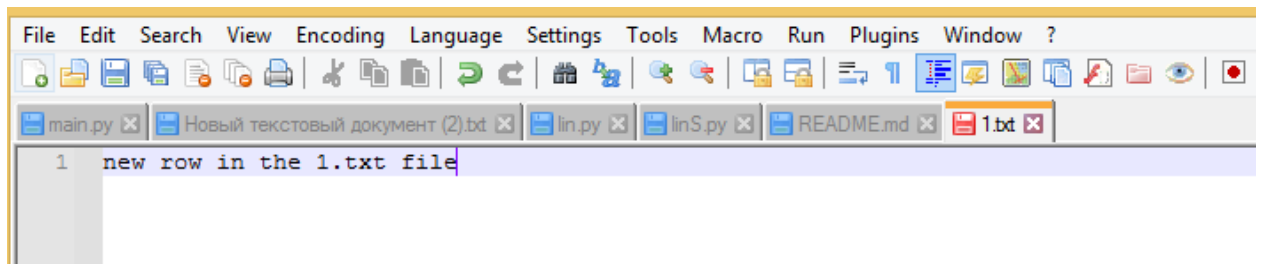


Рисунок 13 - Изменённый текстовый файл 1.txt

```
Student@PC-03-9_521 MINGW64 ~/Desktop/Новая папка/lab1.3 (new_branch)
$ git add 1.txt

Student@PC-03-9_521 MINGW64 ~/Desktop/Новая папка/lab1.3 (new_branch)
$ git commit -m "new row in the 1.txt file"
[new_branch 85bd4b7] new row in the 1.txt file
1 file changed, 1 insertion(+)

Student@PC-03-9_521 MINGW64 ~/Desktop/Новая папка/lab1.3 (new_branch)
$ |
```

Рисунок 14 – Индексируем файл 1.txt и фиксируем его

12. Переходим на ветку main и сливаем ново созданные ветки с main

```
leo@DESKTOP-PBJVSEM MINGW64 ~/Desktop/git/lab1.3 (main)
$ git merge my_first_branch
Updating 954a9ea..16ef6ef
Fast-forward
 in_branch.txt | 0
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 in_branch.txt

leo@DESKTOP-PBJVSEM MINGW64 ~/Desktop/git/lab1.3 (main)
$ git merge new_branch
Merge made by the 'ort' strategy.
1.txt | 1 +
1 file changed, 1 insertion(+)
```

Рисунок 14 – Слияние веток

13. Удаляем ветки my_first_branch и new_branch

```
leo@DESKTOP-PBJVSEM MINGW64 ~/Desktop/git/lab1.3 (main)
$ git branch -d my_first_branch
Deleted branch my_first_branch (was 16ef6ef).

leo@DESKTOP-PBJVSEM MINGW64 ~/Desktop/git/lab1.3 (main)
$ git branch -d new_branch
Deleted branch new_branch (was 464a828).
```

Рисунок 15 – Удаление веток

14. Создаем новые ветки (branch_1 и branch_2)

```

Leo@DESKTOP-PBJVSEM MINGW64 ~/Desktop/git/lab1.3 (main)
$ git branch branch_1

Leo@DESKTOP-PBJVSEM MINGW64 ~/Desktop/git/lab1.3 (main)
$ git branch branch_2

```

Рисунок 16 – Создание веток

15. Перейдём на ветку branch_1 и изменим файл 1.txt файл 3.txt и КОММИТИМ ИЗМЕНЕНИЯ

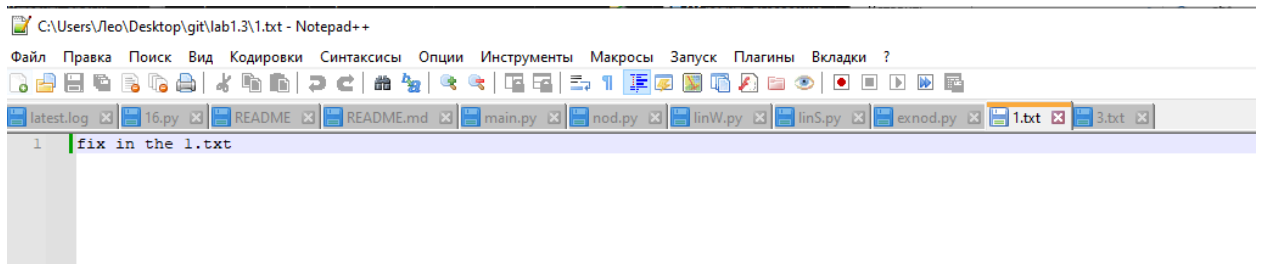


Рисунок 17 – Файл 1.txt в branch_1

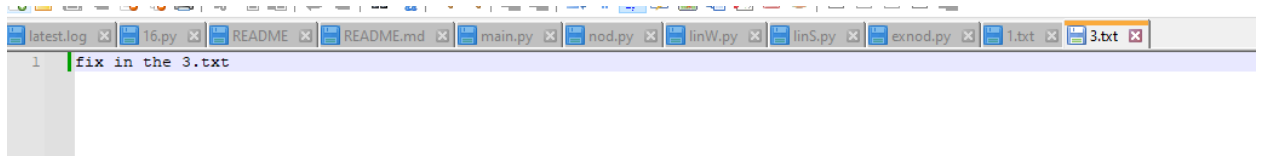


Рисунок 18 – Файл 3.txt в branch_1

16. Перейдём на ветку branch_2 и изменим файл 1.txt файл 3.txt и КОММИТИМ ИЗМЕНЕНИЯ

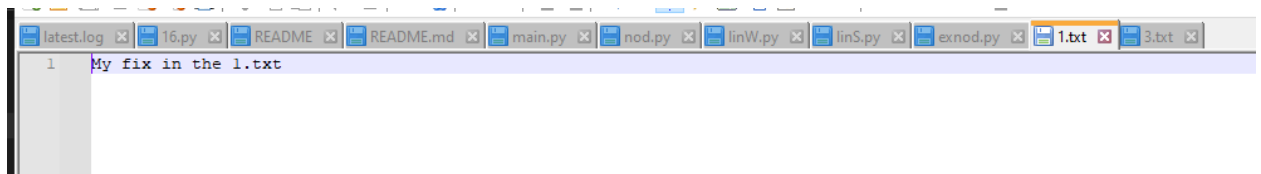


Рисунок 19 – Файл 1.txt в branch_2

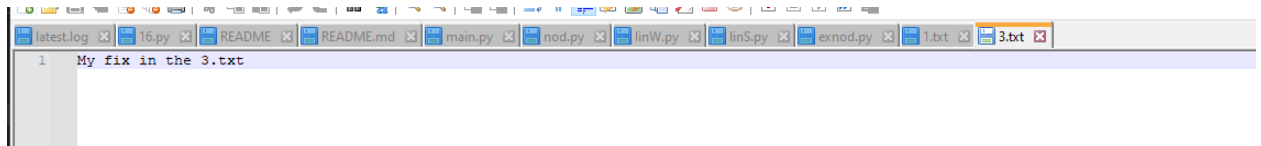


Рисунок 20 – Файл 3.txt в branch_2

```

leo@DESKTOP-PBJVSEM MINGW64 ~/Desktop/git/lab1.3 (branch_1)
$ git commit -m "redact 1.txt and 3.txt"
[branch_1 e6de45c] redact 1.txt and 3.txt
2 files changed, 2 insertions(+), 1 deletion(-)

leo@DESKTOP-PBJVSEM MINGW64 ~/Desktop/git/lab1.3 (branch_1)
$ git checkout branch_2
Switched to branch 'branch_2'

leo@DESKTOP-PBJVSEM MINGW64 ~/Desktop/git/lab1.3 (branch_2)
$ git add 1.txt

leo@DESKTOP-PBJVSEM MINGW64 ~/Desktop/git/lab1.3 (branch_2)
$ git add 3.txt

leo@DESKTOP-PBJVSEM MINGW64 ~/Desktop/git/lab1.3 (branch_2)
$ git commit -m "redact 1.txt and 3.txt"
[branch_2 14c4256] redact 1.txt and 3.txt
2 files changed, 2 insertions(+), 1 deletion(-)

leo@DESKTOP-PBJVSEM MINGW64 ~/Desktop/git/lab1.3 (branch_2)
$

```

Рисунок 21 – Коммит изменений

17. Сливаем ветки branch_1 и branch_2

```

leo@DESKTOP-PBJVSEM MINGW64 ~/Desktop/git/lab1.3 (branch_1)
$ git merge branch_2
Auto-merging 1.txt
CONFLICT (content): Merge conflict in 1.txt
Auto-merging 3.txt
CONFLICT (content): Merge conflict in 3.txt
Automatic merge failed; fix conflicts and then commit the result.

```

Рисунок 22 – Конфликт слияния

18. Решение конфликта: 1 – ручной, 2 – при помощи команды git mergetool.

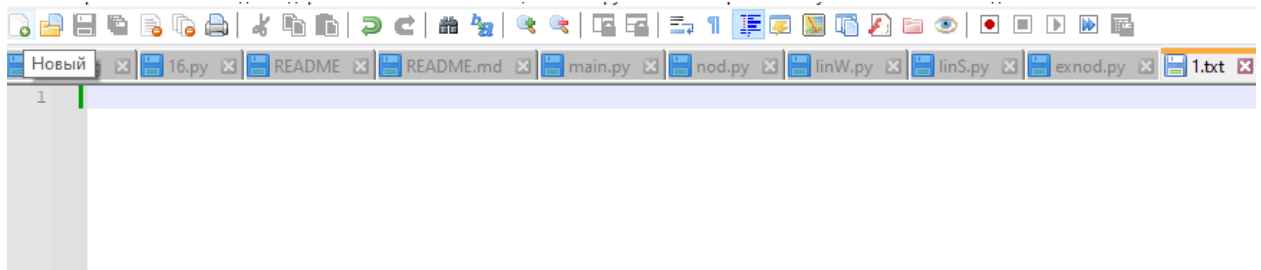


Рисунок 23 – Ручное изменения файла 1.txt

```

fatal: Exiting because of an unresolved conflict.
U      3.txt

```

Рисунок 24 – Конфликт в 3.txt

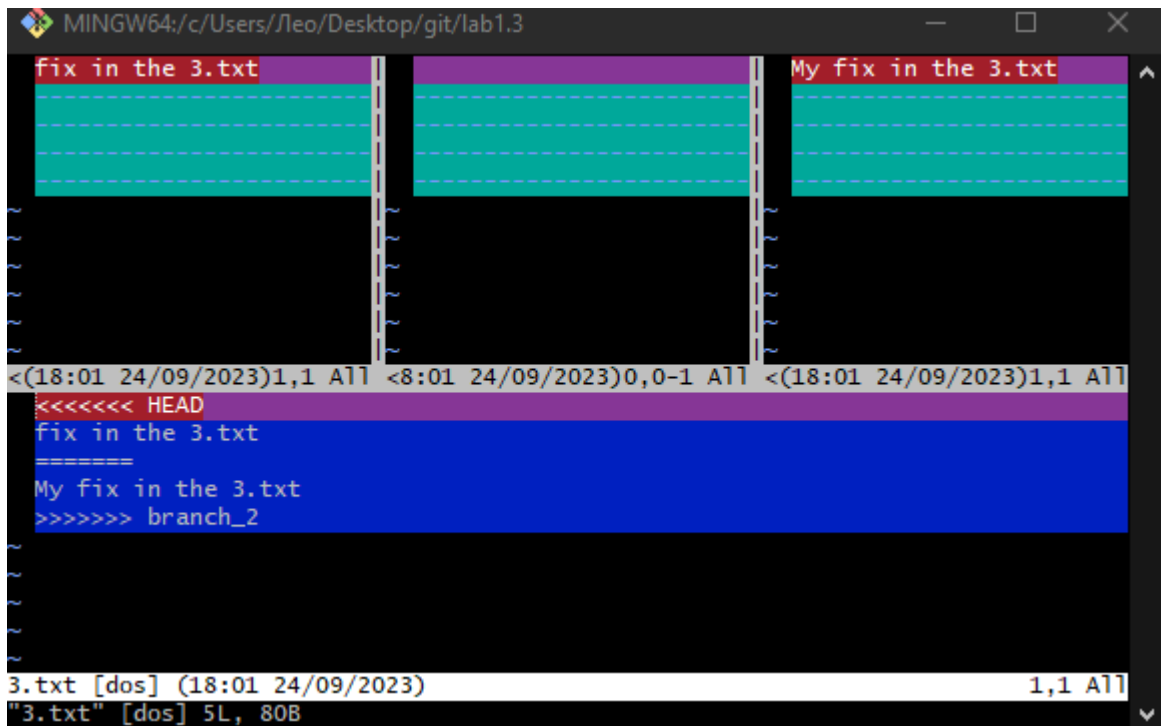


Рисунок 25 – Meld

```
leo@DESKTOP-PBJVSEM MINGW64 ~/Desktop/git/lab1.3 (branch_1|MERGING)
$ git commit -m "3.txt"
[branch_1 c555472] 3.txt

leo@DESKTOP-PBJVSEM MINGW64 ~/Desktop/git/lab1.3 (branch_1)
$ git merge branch_2
Already up to date.
```

Рисунок 26 – Успешное слияние

19. Отправляем ветку на github

```
Лео@DESKTOP-PBJVSEM MINGW64 ~/Desktop/git/lab1.3 (branch_1)
$ git push origin branch_1
Enumerating objects: 20, done.
Counting objects: 100% (20/20), done.
Delta compression using up to 12 threads
Compressing objects: 100% (13/13), done.
Writing objects: 100% (19/19), 1.63 KiB | 837.00 KiB/s, done.
Total 19 (delta 7), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (7/7), done.
remote:
remote: Create a pull request for 'branch_1' on GitHub by visiting:
remote:   https://github.com/FiaLDI/lab1.3/pull/new/branch_1
remote:
To https://github.com/FiaLDI/lab1.3.git
 * [new branch]      branch_1 -> branch_1

Лео@DESKTOP-PBJVSEM MINGW64 ~/Desktop/git/lab1.3 (branch_1)
$
```

Рисунок 27 – Отправка ветки на удалённый репозиторий

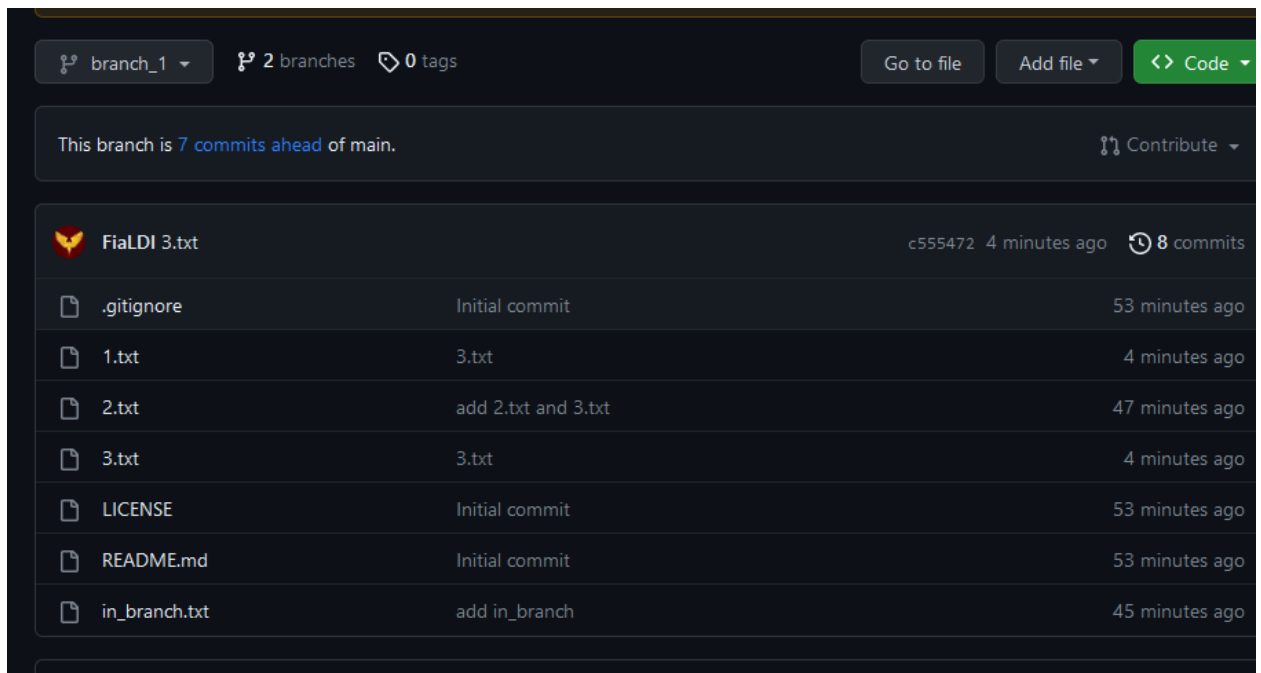


Рисунок 28 – Ветка на GitHub

20. Создаём средствами GitHub ветку brunch_3 и редактируем файл 2.txt

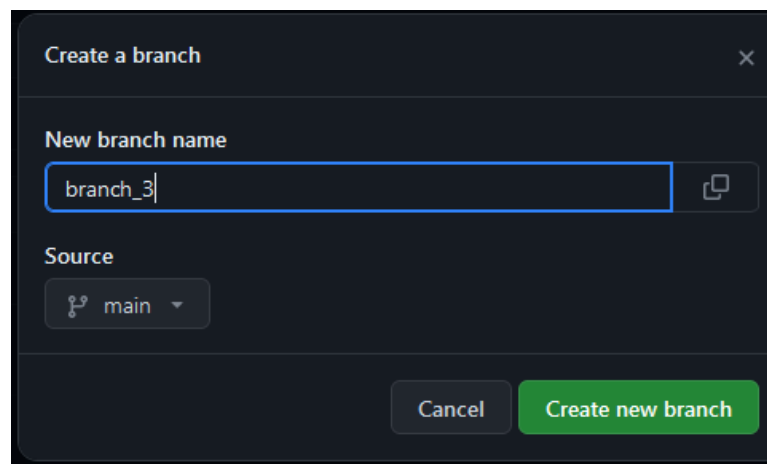


Рисунок 29 – Ввод имени ветки

21. Создаём в локальном репозитории ветку отслеживанияч

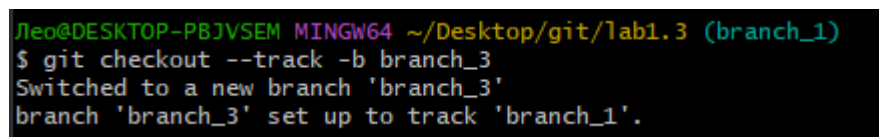


Рисунок 30 – Создание отслеживаемой ветки

22. Переходим в ветку brunch_3 и изменяем файл 2.txt

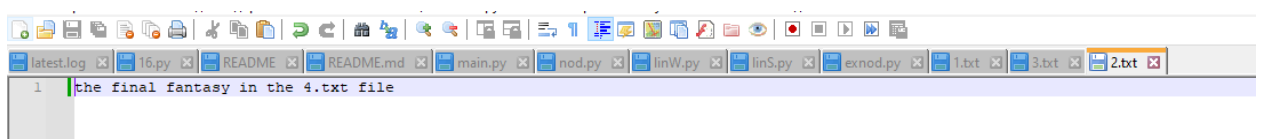


Рисунок 31 – Изменения в файле 2.txt

```

leo@DESKTOP-PBJVSEM MINGW64 ~/Desktop/git/lab1.3 (branch_3)
$ git add 2.txt

leo@DESKTOP-PBJVSEM MINGW64 ~/Desktop/git/lab1.3 (branch_3)
$ git commit -m "redact 2.txt"
[branch_3 3032ca7] redact 2.txt
1 file changed, 1 insertion(+)

```

Рисунок 32 – Фиксация изменений

23. Перемещаю ветки:

```

leo@DESKTOP-PBJVSEM MINGW64 ~/Desktop/git/lab1.3 (main)
$ git rebase branch_2
Successfully rebased and updated refs/heads/main.

```

Рисунок 33 – Перемещение веток

```

leo@DESKTOP-PBJVSEM MINGW64 ~/Desktop/git/lab1.3 (branch_2)
$ git push origin branch_2
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 259 bytes | 259.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/FiaLDI/lab1.3.git
a423528..6bddff8 branch_2 -> branch_2

```

Рисунок 34 – Отправка ветки branch_2

Вывод: исследовали базовые возможности по работе с локальными и удалёнными ветками Git

Ответы на контрольные вопросы:

1. Ветка — это механизм, который позволяет вам работать с разными версиями вашего проекта. Когда вы создаете новую ветку, вы создаете отдельную линию разработки, в которой вы можете вносить изменения без влияния на основную ветку проекта. Это позволяет вам экспериментировать, исправлять ошибки и добавлять новые функции, не затрагивая основную работу.

2. HEAD — это указатель на текущую активную ветку в вашем репозитории. Он указывает на последний коммит в текущей ветке и является отправной точкой для выполнения операций с Git.

3. Создание ветки: `Git branch <name>`

4. Текущая ветка: `git branch`

5. Переключение на ветку: `git checkout <branch_name>`

6. Удаленная ветка в Git – это ветка, которая существует в удаленном репозитории. Она представляет собой версию вашего проекта, которая хранится на удаленном сервере, и может быть доступна для совместной работы и синхронизации с другими разработчиками.

7. Ветка отслеживания в Git – это локальная ветка, которая связана с удаленной веткой в удаленном репозитории. Она отслеживает изменения в удаленной ветке и автоматически обновляется при выполнении операций, таких как `git fetch` или `git pull`.

8. Создание ветки отслеживания: `git checkout -b <branch_name> <remote_name>/<remote_branch_name>`. Где `<branch_name>` - имя новой локальной ветки, `<remote_name>` - имя удаленного репозитория и `<remote_branch_name>` - имя удаленной ветки, которую вы хотите отслеживать.

9. Отправка изменений из локальной в удалённую ветку: `git push <remote_name> <local_branch_name>:<remote_branch_name>`

10. Команды `git fetch` и `git pull` используются для получения изменений из удаленного репозитория в локальный репозиторий в Git. Однако, есть некоторые отличия в их функциональности:

1) `git fetch`: Команда `git fetch` загружает все изменения из удаленного репозитория в локальный репозиторий, но не автоматически объединяет их с текущей веткой. Это позволяет вам просмотреть и анализировать полученные изменения перед их объединением. Команда `git fetch` обновляет информацию о ветках и коммитах в удаленном репозитории, но не изменяет вашу текущую рабочую ветку.

2) `git pull`: Команда `git pull` выполняет две операции: сначала она выполняет `git fetch`, чтобы загрузить изменения из удаленного репозитория, а затем автоматически объединяет эти изменения с текущей веткой. Это означает, что `git pull` обновляет вашу текущую рабочую ветку, применяя изменения из удаленной ветки.

11. Локальную: `git branch -d <name>`, Удалённую: `git push origin --delete <name>`

12. Git-flow – это модель ветвления, которая предлагает структуру и набор правил для эффективной организации работы с ветками в Git. Она предназначена для упрощения совместной разработки и управления версиями проекта. Давайте рассмотрим основные типы веток, организацию работы с ветками и недостатки модели Git-flow.

Основные типы веток в модели Git-flow:

1) Master (главная ветка): в этой ветке хранятся стабильные версии продукта. Каждый коммит в этой ветке обычно соответствует выпуску новой версии.

2) Develop (ветка разработки): в этой ветке происходит активная разработка проекта. Она является основной веткой для интеграции всех функциональных изменений.

3) Feature (ветки функциональности): Каждая новая функциональность разрабатывается в отдельной ветке. Они отходят от ветки Develop и объединяются обратно в нее после завершения разработки.

4) Release (ветки релизов): Ветки релизов создаются для подготовки к выпуску новой версии продукта. Они позволяют исправить ошибки, подготовить документацию и выполнить другие задачи, связанные с релизом.

5) Hotfix (ветки исправлений): если в процессе эксплуатации продукта обнаруживается критическая ошибка, создается ветка исправлений. Она позволяет быстро внести исправления и применить их как в текущей версии, так и в будущих релизах.