

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития  
Кафедра инфокоммуникаций

ОТЧЁТ

по лабораторной работе №1.3

**Дисциплина: «Программирование на Python»**  
**Тема: «Основы ветвления Git»**

Выполнил:  
Епифанов Алексей Александрович  
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 Роман Александрович

---

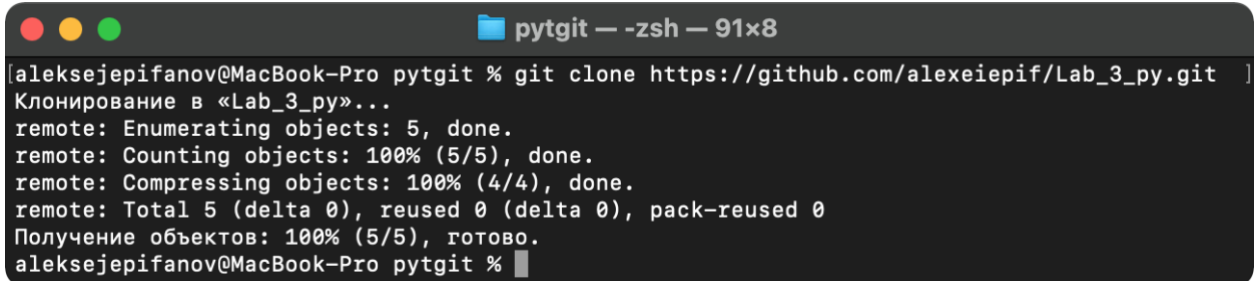
(подпись)

Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_  
Ставрополь, 2023 г.

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

Порядок выполнения работы:

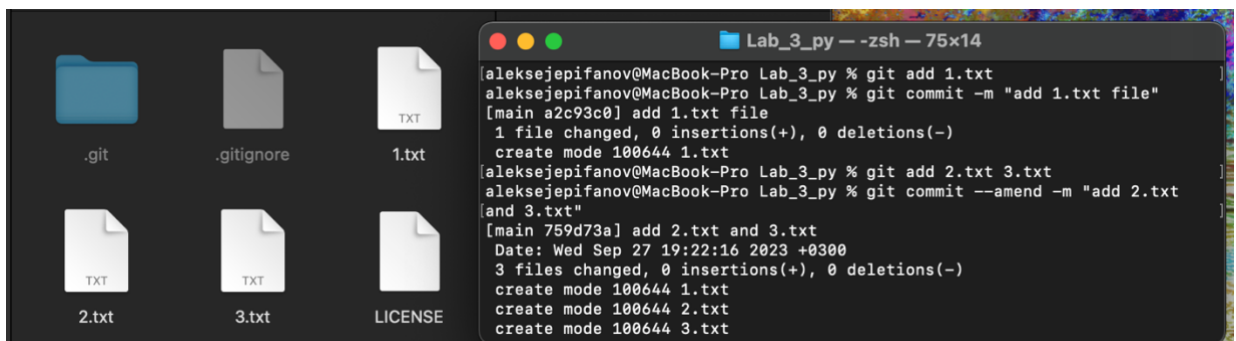
1. Создал новый репозиторий и клонировал его:

A terminal window titled 'pytgit — zsh — 91x8' showing the command 'git clone https://github.com/alexeiepif/Lab\_3\_py.git' and its output. The output shows the cloning process: enumerating objects, counting objects, compressing objects, and finally receiving the objects. The terminal text is as follows:

```
aleksejepifanov@MacBook-Pro pytgit % git clone https://github.com/alexeiepif/Lab_3_py.git
Клонирование в «Lab_3_py»...
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
Получение объектов: 100% (5/5), готово.
aleksejepifanov@MacBook-Pro pytgit %
```

Рисунок 1. Клонирование репозитория

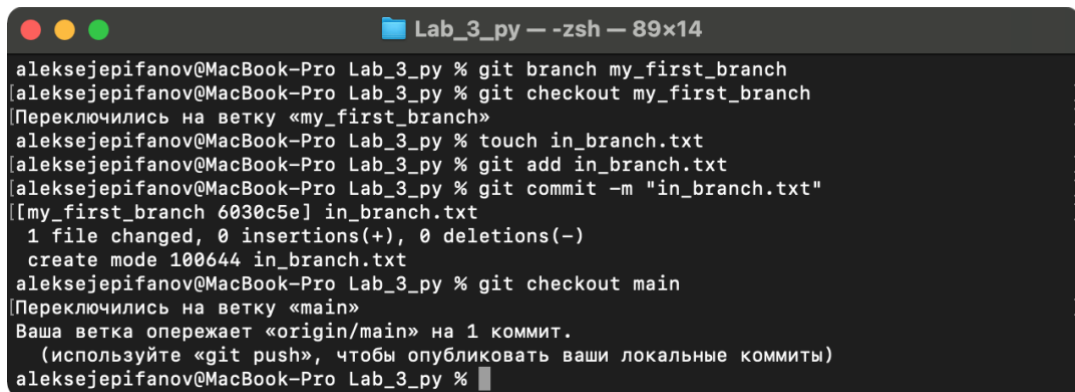
2. Создал 3 файла: 1.txt, 2.txt и 3.txt. Проиндексировал первый файл и сделал коммит, потом проиндексировал оставшиеся и перезаписал уже сделанный коммит с новым именем:

A composite image showing a file explorer window on the left and a terminal window on the right. The file explorer shows a directory with files '.git', '.gitignore', '1.txt', '2.txt', '3.txt', and 'LICENSE'. The terminal window, titled 'Lab\_3\_py — zsh — 75x14', shows the commands to add 1.txt, commit it, add 2.txt and 3.txt, and amend the previous commit. The terminal text is as follows:

```
aleksejepifanov@MacBook-Pro Lab_3_py % git add 1.txt
aleksejepifanov@MacBook-Pro Lab_3_py % git commit -m "add 1.txt file"
[main a2c93c0] add 1.txt file
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 1.txt
aleksejepifanov@MacBook-Pro Lab_3_py % git add 2.txt 3.txt
aleksejepifanov@MacBook-Pro Lab_3_py % git commit --amend -m "add 2.txt
and 3.txt"
[main 759d73a] add 2.txt and 3.txt
Date: Wed Sep 27 19:22:16 2023 +0300
3 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 1.txt
create mode 100644 2.txt
create mode 100644 3.txt
```

Рисунок 2. Перезапись коммита

3. Создал новую Ветку и перешел на нее, после чего создал новый файл, закоммитил изменения и вернулся на ветку main:

A terminal window titled 'Lab\_3\_py — zsh — 89x14' showing the commands to create a new branch, checkout to it, create a file, commit it, and then checkout back to the main branch. The terminal text is as follows:

```
aleksejepifanov@MacBook-Pro Lab_3_py % git branch my_first_branch
aleksejepifanov@MacBook-Pro Lab_3_py % git checkout my_first_branch
Переключились на ветку «my_first_branch»
aleksejepifanov@MacBook-Pro Lab_3_py % touch in_branch.txt
aleksejepifanov@MacBook-Pro Lab_3_py % git add in_branch.txt
aleksejepifanov@MacBook-Pro Lab_3_py % git commit -m "in_branch.txt"
[[my_first_branch 6030c5e] in_branch.txt
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 in_branch.txt
aleksejepifanov@MacBook-Pro Lab_3_py % git checkout main
Переключились на ветку «main»
Ваша ветка опережает «origin/main» на 1 коммит.
(используйте «git push», чтобы опубликовать ваши локальные коммиты)
aleksejepifanov@MacBook-Pro Lab_3_py %
```

Рисунок 3. Создание и перемещение между ветками

4. Создал и сразу же перешел на новую ветку, добавил строку в файл 1.txt, закоммитил изменения:

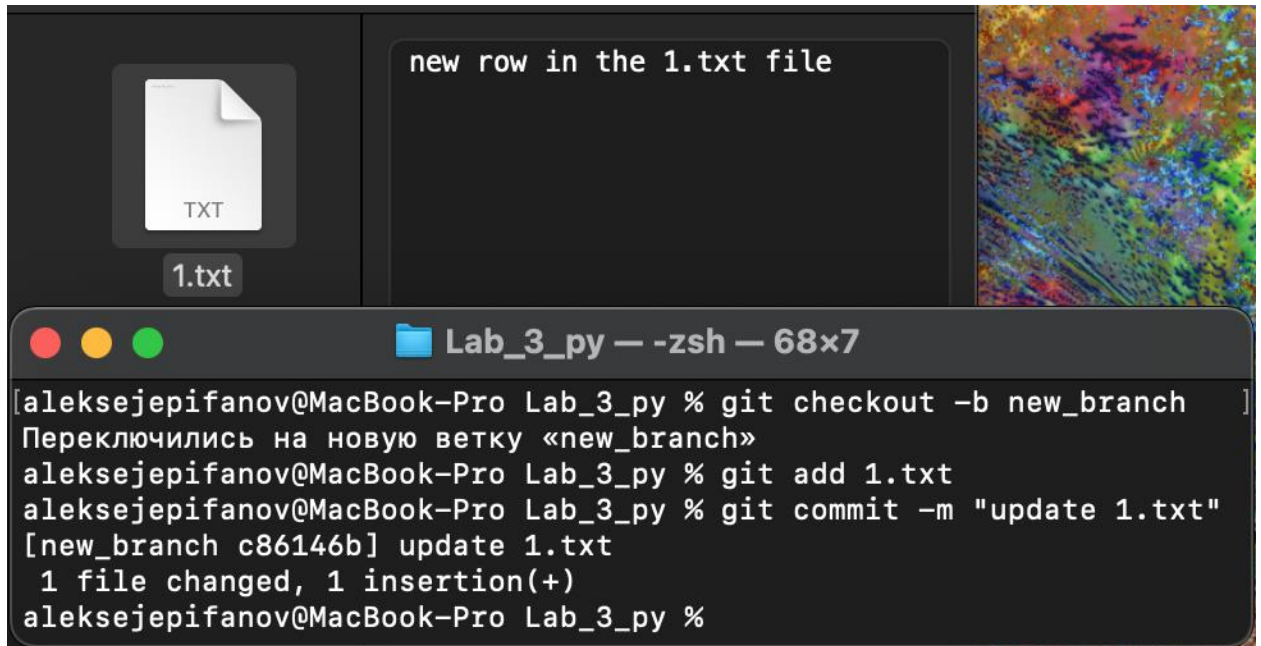


Рисунок 4. Создание новой ветки и коммит обновленного файла

5. Переключился на ветку main, слил ее с my\_first\_branch, но не получилось выполнить слияние с new\_branch:

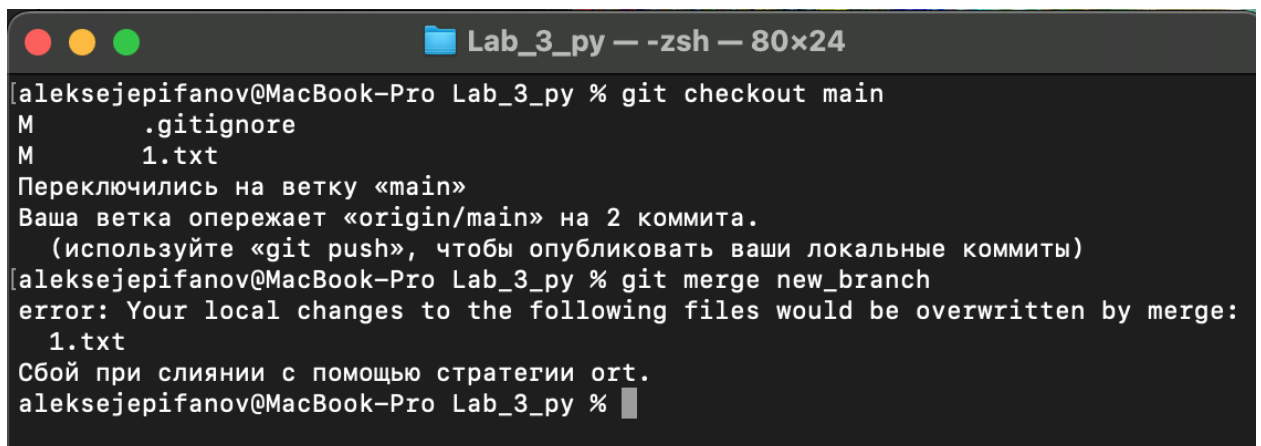


Рисунок 5. Выполнил одно слияние, но не выполнил второе

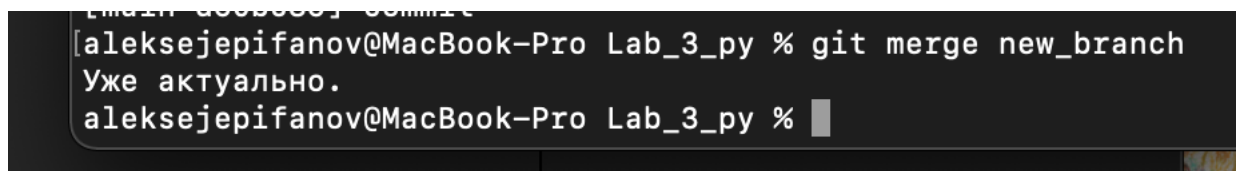
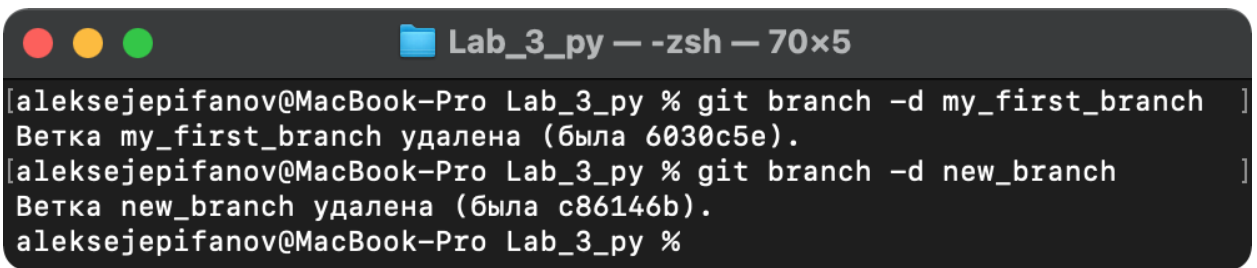


Рисунок 6. Получилось выполнить слияние с new\_branch

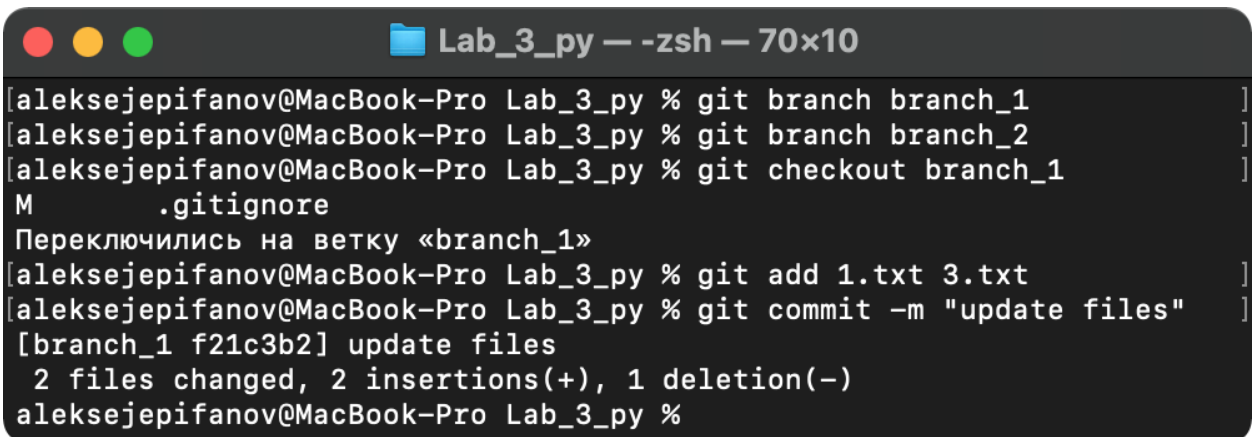
6. Удалил созданные ветки:



```
Lab_3_py — -zsh — 70x5
[aleksejerpifanov@MacBook-Pro Lab_3_py % git branch -d my_first_branch
Ветка my_first_branch удалена (была 6030c5e).
[aleksejerpifanov@MacBook-Pro Lab_3_py % git branch -d new_branch
Ветка new_branch удалена (была c86146b).
aleksejerpifanov@MacBook-Pro Lab_3_py %
```

Рисунок 7. Удаление веток

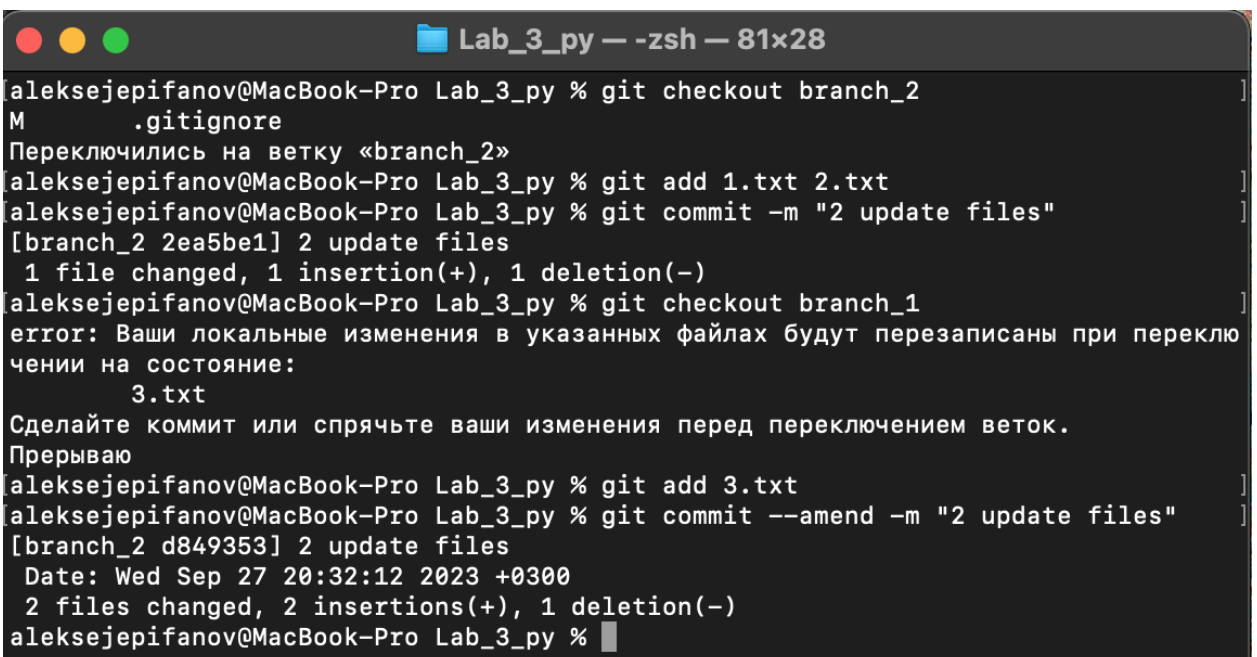
7. Создал новые ветки, перешел в ветку branch\_1 и изменил файлы 1.txt, 3.txt, закоммитил изменения:



```
Lab_3_py — -zsh — 70x10
[aleksejerpifanov@MacBook-Pro Lab_3_py % git branch branch_1
[aleksejerpifanov@MacBook-Pro Lab_3_py % git branch branch_2
[aleksejerpifanov@MacBook-Pro Lab_3_py % git checkout branch_1
М .gitignore
Переключились на ветку «branch_1»
[aleksejerpifanov@MacBook-Pro Lab_3_py % git add 1.txt 3.txt
[aleksejerpifanov@MacBook-Pro Lab_3_py % git commit -m "update files"
[branch_1 f21c3b2] update files
2 files changed, 2 insertions(+), 1 deletion(-)
aleksejerpifanov@MacBook-Pro Lab_3_py %
```

Рисунок 8. Создал новые ветки, закоммитил изменения файлов в первой

8. Перешел во вторую созданную ветку и изменил там те же самые файлы, закоммитил изменения:



```
Lab_3_py — -zsh — 81x28
aleksejerpifanov@MacBook-Pro Lab_3_py % git checkout branch_2
М .gitignore
Переключились на ветку «branch_2»
aleksejerpifanov@MacBook-Pro Lab_3_py % git add 1.txt 2.txt
aleksejerpifanov@MacBook-Pro Lab_3_py % git commit -m "2 update files"
[branch_2 2ea5be1] 2 update files
1 file changed, 1 insertion(+), 1 deletion(-)
aleksejerpifanov@MacBook-Pro Lab_3_py % git checkout branch_1
error: Ваши локальные изменения в указанных файлах будут перезаписаны при переключении на состояние:
3.txt
Сделайте коммит или спрячьте ваши изменения перед переключением веток.
Прерываю
aleksejerpifanov@MacBook-Pro Lab_3_py % git add 3.txt
aleksejerpifanov@MacBook-Pro Lab_3_py % git commit --amend -m "2 update files"
[branch_2 d849353] 2 update files
Date: Wed Sep 27 20:32:12 2023 +0300
2 files changed, 2 insertions(+), 1 deletion(-)
aleksejerpifanov@MacBook-Pro Lab_3_py %
```

Рисунок 9. Закоммитил новые изменения в файлах в ветке branch\_2

9. Перешел в ветку branch\_1 и попытался слить в нее branch\_2:

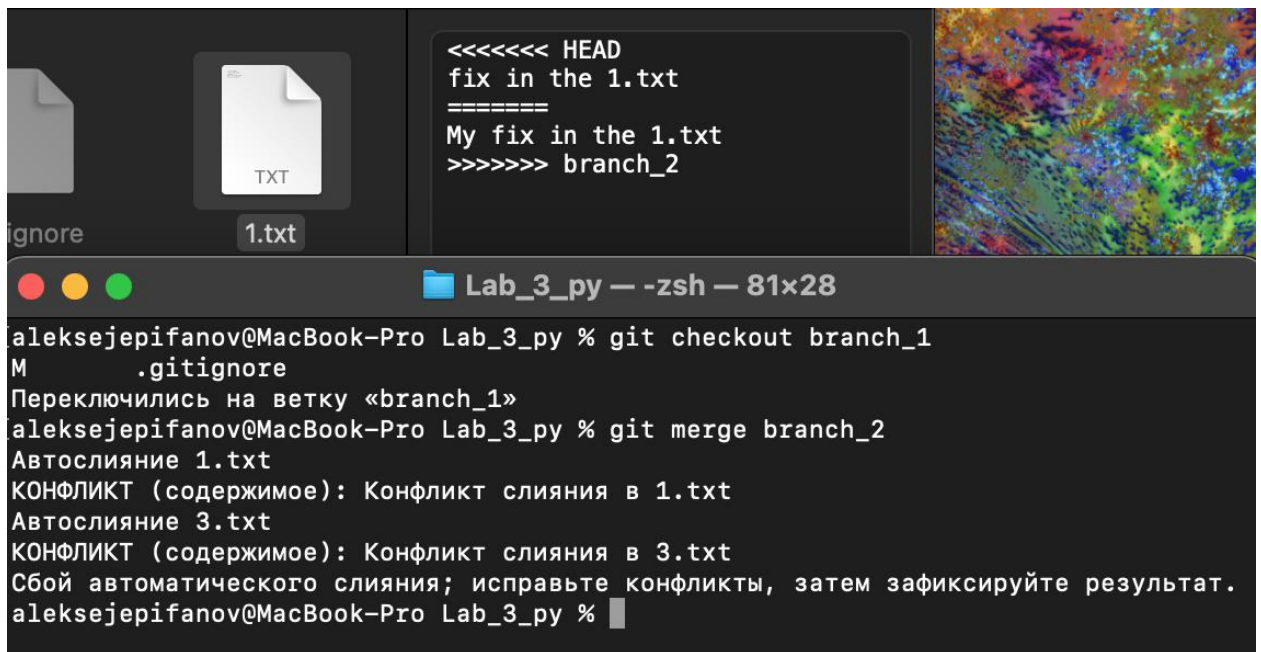


Рисунок 10. Попытка слияния завершилась неудачно

10. Исправил конфликты, один вручную: изменил файл, другой используя git mergetool:

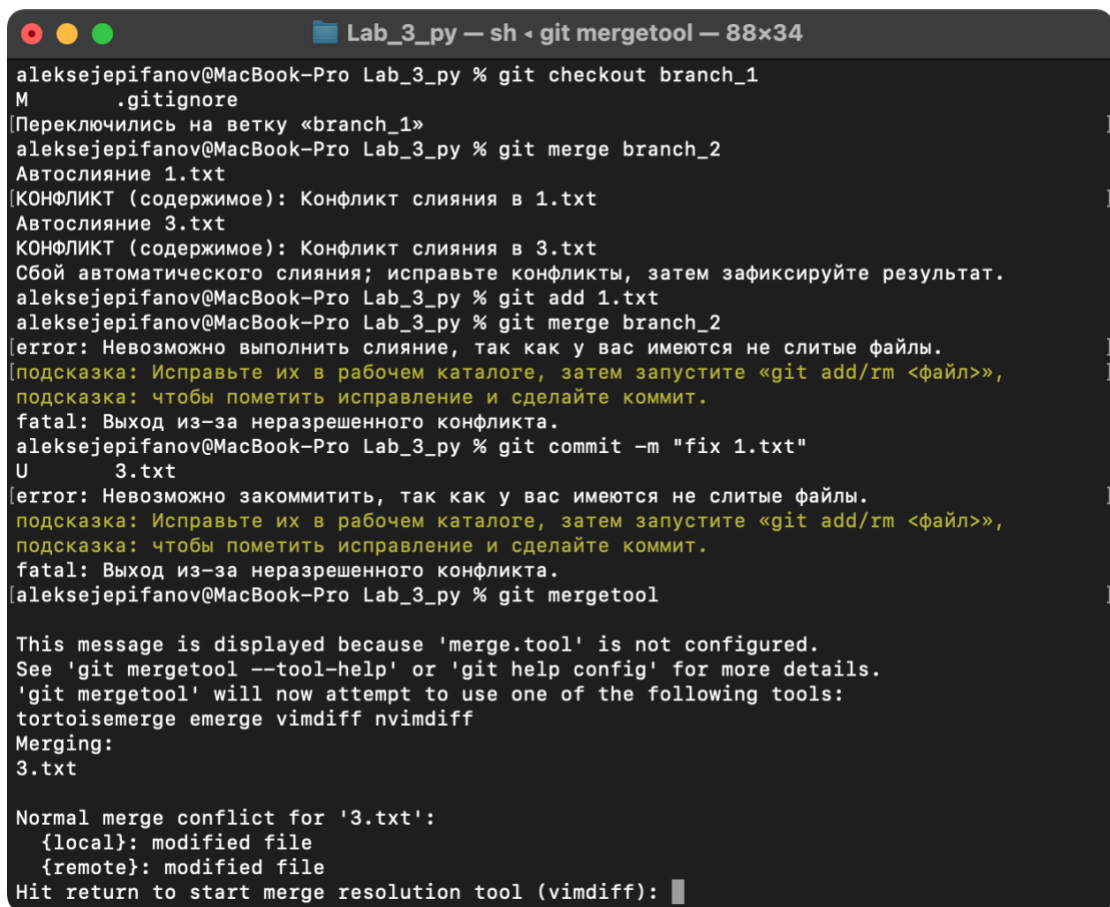


Рисунок 11. 1.txt я исправил вручную, далее вызвал команду git mergetool

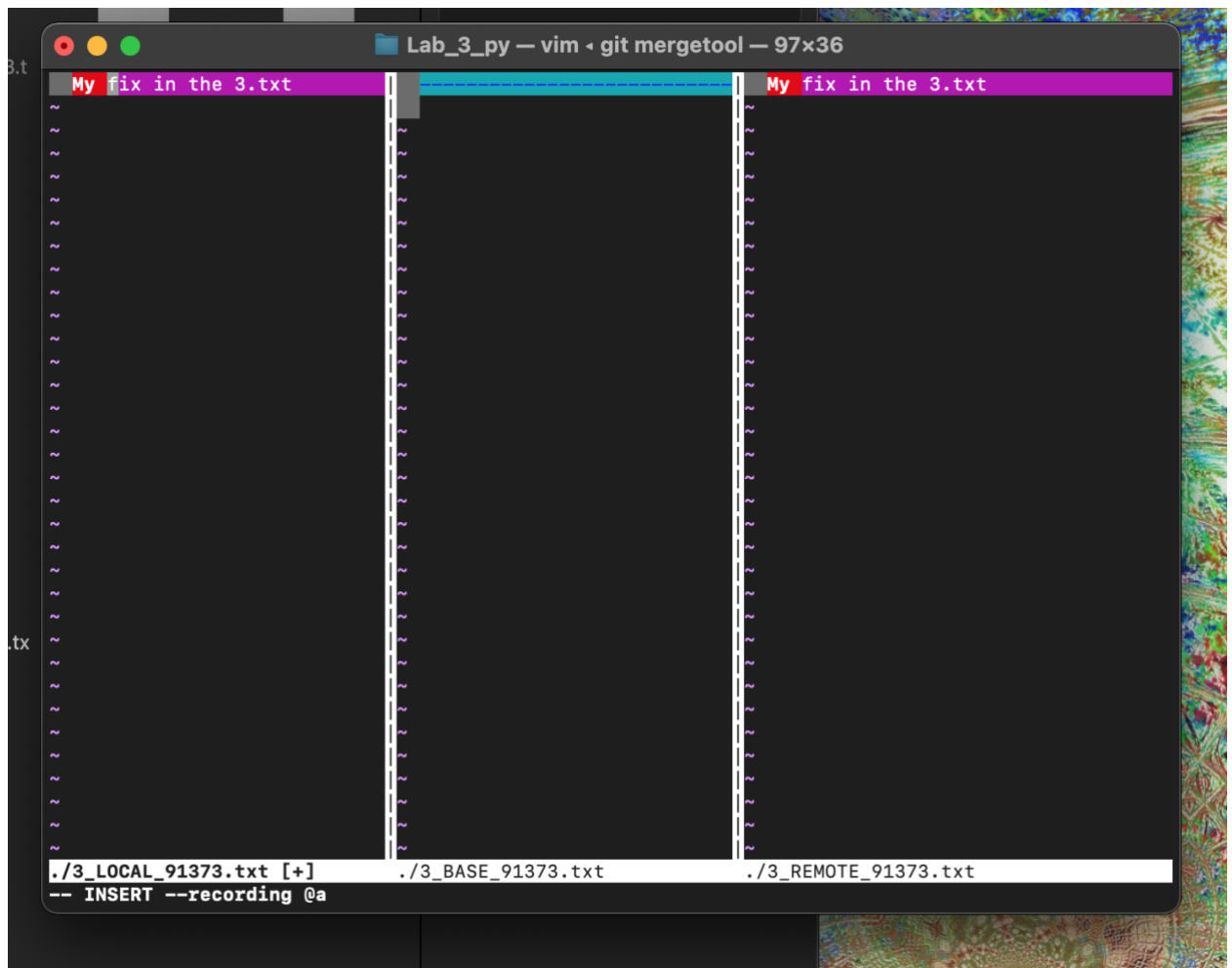


Рисунок 12. Инструмент tortoisemerge

```

[aleksejeripifanov@MacBook-Pro Lab_3_py % git merge branch_2
fatal: Вы не завершили слияние (присутствует файл MERGE_HEAD).
Выполните коммит ваших изменений, перед слиянием.
[aleksejeripifanov@MacBook-Pro Lab_3_py % git commit -m "committt"
[branch_1 423d473] committt
[aleksejeripifanov@MacBook-Pro Lab_3_py % git merge branch_2
Уже актуально.
[aleksejeripifanov@MacBook-Pro Lab_3_py %

```

Рисунок 13. Завершил слияние

11. Отправил ветку branch\_1 на удаленный репозиторий:



```
Lab_3_py — zsh — 99x15
aleksejefifanov@MacBook-Pro Lab_3_py % git push origin branch_1
Перечисление объектов: 23, готово.
Подсчет объектов: 100% (23/23), готово.
При сжатии изменений используется до 8 потоков
Сжатие объектов: 100% (14/14), готово.
Запись объектов: 100% (22/22), 1.72 КиБ | 880.00 КиБ/с, готово.
Всего 22 (изменений 8), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (8/8), done.
remote:
remote: Create a pull request for 'branch_1' on GitHub by visiting:
remote:   https://github.com/alexeiepif/Lab_3_py/pull/new/branch_1
remote:
To https://github.com/alexeiepif/Lab_3_py.git
 * [new branch]      branch_1 -> branch_1
aleksejefifanov@MacBook-Pro Lab_3_py %
```

Рисунок 14. Отправка ветки

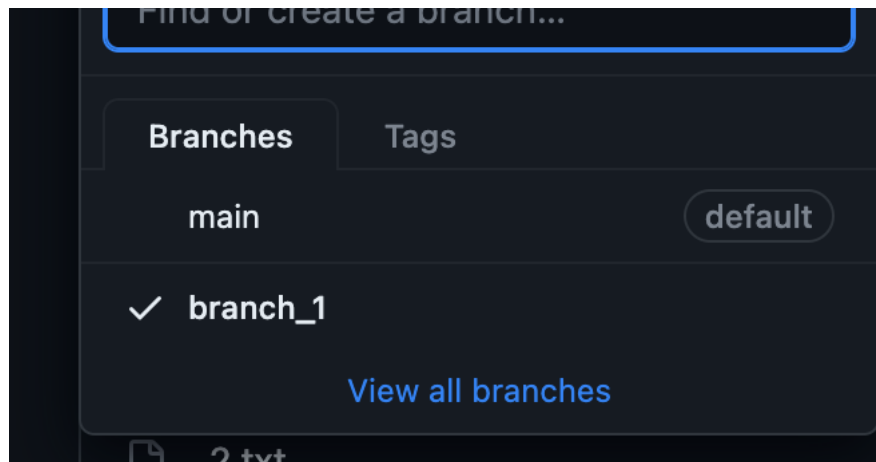


Рисунок 15. Ветки в GitHub

12. Создал ветку средствами GitHub и перенес ее в локальный репозиторий:

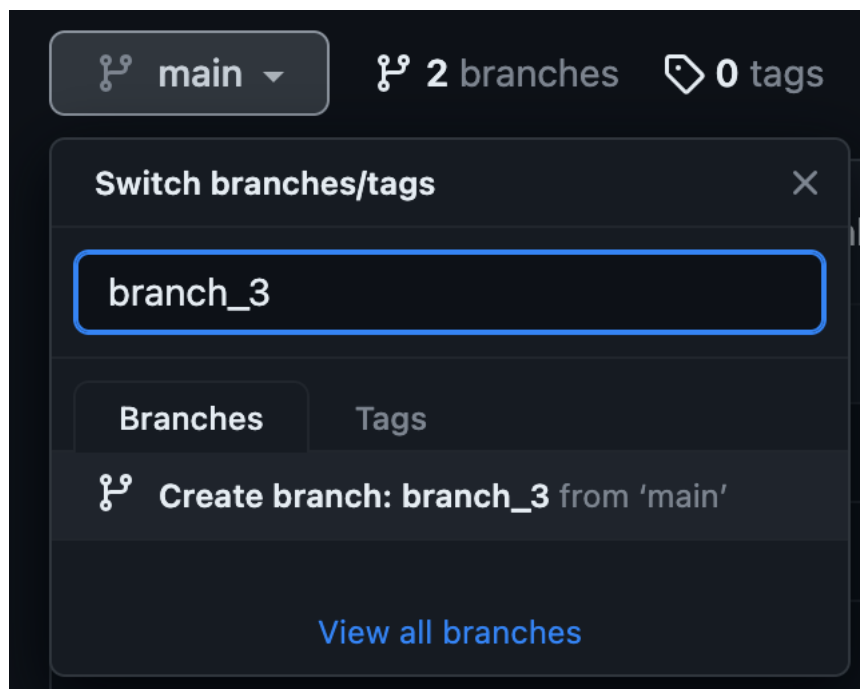


Рисунок 16. Создание ветки

```
Lab_3_py — -zsh — 80x13
[aleksejefifanov@MacBook-Pro Lab_3_py % git pull
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Распаковка объектов: 100% (1/1), 623 байта | 623.00 КиБ/с, готово.
Из https://github.com/alexeiefif/Lab_3_py
e2713fe..5a7e2c0  main    -> origin/main
* [новая ветка]    branch_3 -> origin/branch_3
Успешно перемещён и обновлён refs/heads/main.
[aleksejefifanov@MacBook-Pro Lab_3_py % git checkout branch_3
branch 'branch_3' set up to track 'origin/branch_3'.
Переключились на новую ветку «branch_3»
aleksejefifanov@MacBook-Pro Lab_3_py %
```

Рисунок 17. Успешный перенос ветки

13. Добавил в файл 2.txt новую строку и закоммитил:

```
[aleksejefifanov@MacBook-Pro Lab_3_py % git add 2.txt
[aleksejefifanov@MacBook-Pro Lab_3_py % git commit -m "update 2.txt"
[branch_3 6a5a21a] update 2.txt
1 file changed, 1 insertion(+)
aleksejefifanov@MacBook-Pro Lab_3_py %
```

Рисунок 18. Коммит файла 2.txt

14. Выполнил перемещение ветки main на ветку banch\_2 и отправил изменения обоих веток на GitHub:

```
Lab_3_py — -zsh — 95x31
Last login: Thu Sep 28 14:34:22 on ttys006
[aleksejefifanov@MacBook-Pro Lab_3_py % git checkout branch_2
Переключились на ветку «branch_2»
[aleksejefifanov@MacBook-Pro Lab_3_py % git rebase main
Успешно перемещён и обновлён refs/heads/branch_2.
[aleksejefifanov@MacBook-Pro Lab_3_py % git checkout main
Переключились на ветку «main»
Ваша ветка опережает «origin/main» на 1 коммит.
(используйте «git push», чтобы опубликовать ваши локальные коммиты)
[aleksejefifanov@MacBook-Pro Lab_3_py % git push origin main
Перечисление объектов: 5, готово.
Подсчет объектов: 100% (5/5), готово.
При сжатии изменений используется до 8 потоков
Сжатие объектов: 100% (3/3), готово.
Запись объектов: 100% (3/3), 298 байтов | 298.00 КиБ/с, готово.
Всего 3 (изменений 2), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/alexeiefif/Lab_3_py.git
5a7e2c0..fa1d170  main -> main
[aleksejefifanov@MacBook-Pro Lab_3_py % git push origin branch_2
error: src refspec branch_2 ничего не соответствует
error: не удалось отправить некоторые ссылки в «https://github.com/alexeiefif/Lab_3_py.git»
[aleksejefifanov@MacBook-Pro Lab_3_py % git push origin branch_2
Всего 0 (изменений 0), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote:
remote: Create a pull request for 'branch_2' on GitHub by visiting:
remote:   https://github.com/alexeiefif/Lab_3_py/pull/new/branch_2
remote:
To https://github.com/alexeiefif/Lab_3_py.git
* [new branch]    branch_2 -> branch_2
aleksejefifanov@MacBook-Pro Lab_3_py %
```

Рисунок 19. Перенос ветки и отправка изменений



## Ответы на контрольные вопросы:

### 1. Что такое ветка?

Ветка в Git — это простой перемещаемый указатель на один из коммитов.

### 2. Что такое HEAD?

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, относительно которого будет создана рабочая копия во время операции checkout.

### 3. Способы создания веток.

Чтобы создать новую ветку, необходимо использовать команду `git branch`.

Чтобы создать ветку и сразу переключиться на нее, можно использовать команду `git checkout -b`.

### 4. Как узнать текущую ветку?

Увидеть текущую ветку можно при помощи простой команды `git log`, которая покажет куда указывают указатели веток. Эта опция называется `--decorate`. HEAD будет стоять рядом с текущей веткой. Также можно использовать `git branch -v`, рядом с текущей веткой будет значок `*`.

### 5. Как переключаться между ветками?

Для переключения на существующую ветку выполните команду `git checkout`.

### 6. Что такое удаленная ветка?

Удалённые ссылки — это ссылки (указатели) в ваших удалённых репозиториях, включая ветки, теги и так далее. Полный список удалённых ссылок можно получить с помощью команды `git ls-remote <remote>` или

команды `git remote show <remote>` для получения удалённых веток и дополнительной информации.

#### 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 удалённым репозиторием, чтобы гарантировать точное соответствие с ним.

#### 8. Как создать ветку отслеживания?

При клонировании репозитория, как правило, автоматически создаётся ветка `master`, которая следит за `origin/master`. Однако, при желании можно настроить отслеживание и других веток — следить за ветками на других серверах или отключить слежение за веткой `master`. Сделать это можно с помощью команды `git checkout -b <branch> <remote>/<branch>`. Это часто используемая команда, поэтому Git предоставляет сокращённую форму записи в виде флага `--track`.

#### 9. Как отправить изменения из локальной ветки в удалённую ветку?

Когда вы хотите поделиться веткой, вам необходимо отправить её на удалённый сервер, где у вас есть права на запись. Ваши локальные ветки автоматически не синхронизируются с удалёнными при отправке — вам нужно явно указать те ветки, которые вы хотите отправить.

Таким образом, вы можете использовать свои личные ветки для работы, которую не хотите показывать, а отправлять только те тематические ветки, над которыми вы хотите работать с кем-то совместно. Отправка осуществляется командой `git push <remote> <branch>`.

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

Команда `git fetch` получает с сервера все изменения, которых у вас ещё нет, но не будет изменять состояние вашей рабочей директории. Эта команда просто получает данные и позволяет вам самостоятельно сделать слияние. Тем не менее, существует команда `git pull`, которая в большинстве случаев является командой `git fetch`, за которой непосредственно следует команда `git merge`.

Если у вас настроена ветка слежения, или она явно установлена, или она была создана автоматически командами `clone` или `checkout`, `git pull` определит сервер и ветку, за которыми следит ваша текущая ветка, получит данные с этого сервера и затем попытается слить удалённую ветку.

#### 11. Как удалить локальную и удалённую ветки?

Для удаления локальной ветки выполните команду `git branch` с параметром `-d`.

Вы можете удалить ветку на удалённом сервере используя параметр `--delete` для команды `git push`.

12. Изучить модель ветвления `git-flow` (использовать материалы статей <https://www.atlassian.com/ru/git/tutorials/comparing-workflows/gitflow-workflow>, <https://habr.com/ru/post/106912/>). Какие основные типы веток присутствуют в модели `git-flow`? Как организована работа с ветками в модели `git-flow`? В чем недостатки `git-flow`?

В модели ветвления `git-flow`, основные типы веток это:

`master` - главная ветка проекта, которая содержит только стабильный код и используется для создания релизов.

`develop` - ветка, в которой ведется основная разработка проекта. Она содержит все изменения, которые были сделаны разработчиками.

`feature` - ветки для добавления новой функциональности в проект. Они ветвятся от ветки `develop`, и после завершения работы ветки объединяются с `develop` веткой.

`release` - ветки для подготовки новой версии проекта. Они ветвятся от ветки `develop`, содержат минимальный набор изменений и используются для подготовки релиза. После тестирования и отладки, ветка объединяется с веткой `master` и `develop`.

`hotfix` - ветки для исправления критических ошибок на производственной версии проекта. Они ветвятся от ветки `master`, и после исправления ошибок объединяются с `master` и `develop` ветками.

Работа с ветками в модели `git-flow` организована следующим образом:

Начало разработки новой функциональности начинается с ветвления от ветки `develop` ветки `feature`, на которой работает разработчик.

После завершения работы, все изменения ветки `feature` тестируются, затем вливаются обратно в ветку `develop`.

В момент подготовки новой версии программного продукта ветка `release` создается из `develop`, и она используется для проведения основных тестов наиболее важных функций.

Ветка `hotfix` создается из ветки `master`, если в производственной версии обнаружена критическая ошибка, и на этой ветке выполняется исправление.

Недостатки `git-flow` включают в себя:

Сложность и необходимость управления множеством веток, что может быть трудным для маленьких команд.

Не слишком хорошо подходит для быстрой разработки и быстрой реализации необходимых исправлений или функций в связи с наличием большого количества ветвлений.

Накладывает значительный набор процедур и правил для разработки и управления релизами.

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.

В Git клиенте `codeberg.org` присутствует возможность работы с ветками Git. Инструменты для работы с ветками в `codeberg.org` соответствуют стандартным инструментам Git и включают в себя:

Создание новой ветки из текущей ветки.

Переключение между ветками.

Объединение веток (с помощью команды `merge`), которые могут быть выполнены из интерфейса Git клиента.

Возможность создавать теги и выполнять релизы (включая создание новой ветки для релиза и склеивание веток для внесения исправлений в производственную версию приложения).

Вывод: в результате выполнения работы были исследованы базовые возможности по работе с локальными и удаленными ветками Git.