

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:
Кулешов Олег Иванович
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, 2023

Тема: Основы ветвления Git

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




 1.txt	Пн 23.10.23 18:24	Текстовый докум...	0 КБ
 2.txt	Пн 23.10.23 18:24	Текстовый докум...	0 КБ
 3.txt	Пн 23.10.23 18:24	Текстовый докум...	0 КБ

Рисунок 1. Создание трёх текстовых файлов

```
(venv) PS C:\Users\User\PycharmProjects\Python_laba_3> git commit --amend -m 'add 2.txt and 3.txt files'
[master 763569b] add 2.txt and 3.txt files
Date: Mon Oct 23 18:31:13 2023 +0300
3 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 1.txt
create mode 100644 2.txt
create mode 100644 3.txt
(venv) PS C:\Users\User\PycharmProjects\Python_laba_3>
```

Рисунок 2. Перезапись коммита

```
(venv) PS C:\Users\User\PycharmProjects\Python_laba_3> git branch my_first_branch
(venv) PS C:\Users\User\PycharmProjects\Python_laba_3> git checkout my_first_branch
```

Рисунок 3. Создание ветки “my_first_branch” и переход на неё

```
(venv) PS C:\Users\User\PycharmProjects\Python_laba_3> git add in_branch.txt
(venv) PS C:\Users\User\PycharmProjects\Python_laba_3> git commit -m 'add in_branch.txt file'
[my_first_branch b3c01bd] add in_branch.txt file
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 in_branch.txt
```

Рисунок 4. Индексация текстового файла “in_branch” и коммит к нему

```
(venv) PS C:\Users\User\PycharmProjects\Python_laba_3> git checkout master
Switched to branch 'master'
```

Рисунок 5. Переключение на ветку “master”

```
(venv) PS C:\Users\User\PycharmProjects\Python_laba_3> git branch new_branch
(venv) PS C:\Users\User\PycharmProjects\Python_laba_3> git checkout new_branch
```

Рисунок 6. Создание ветки “new_branch” и переключение на неё

```
(venv) PS C:\Users\User\PycharmProjects\Python_laba_3> git add 1.txt
(venv) PS C:\Users\User\PycharmProjects\Python_laba_3> git commit -m 'changed in 1.txt'
[new_branch 4dc85a3] changed in 1.txt
1 file changed, 1 insertion(+)
```

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

```
(venv) PS C:\Users\User\PycharmProjects\Python_laba_3> git checkout master
(venv) PS C:\Users\User\PycharmProjects\Python_laba_3> git merge my_first_branch
Updating 763569b..b3c01bd
Fast-forward
 in_branch.txt | 0
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 in_branch.txt
(venv) PS C:\Users\User\PycharmProjects\Python_laba_3> git merge new_branch
Merge made by the 'ort' strategy.
1.txt | 1 +
1 file changed, 1 insertion(+)
```

Рисунок 8. Переключение на ветку “master” и слияние с ветками “my_first_branch” и “new_branch”

```
(venv) PS C:\Users\User\PycharmProjects\Python_laba_3> git branch -d my_first_branch
Deleted branch my_first_branch (was b3c01bd).
(venv) PS C:\Users\User\PycharmProjects\Python_laba_3> git branch -d new_branch
Deleted branch new_branch (was 4dc85a3).
```

Рисунок 9. Удаление веток “my_first_branch” и “new_branch”

```
(venv) PS C:\Users\User\PycharmProjects\Python_laba_3> git branch branch_1
(venv) PS C:\Users\User\PycharmProjects\Python_laba_3> git branch branch_2
```

Рисунок 10. Создание веток “branch_1” и “branch_2”

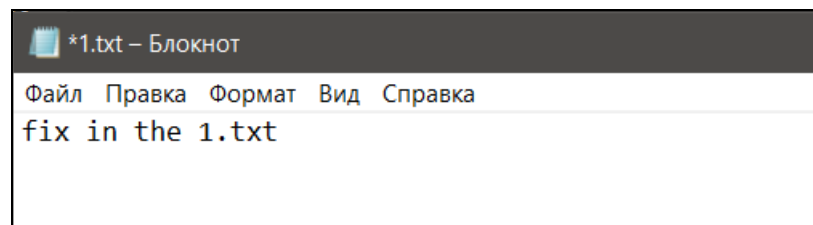


Рисунок 11. Добавил изменение в файл “1.txt”

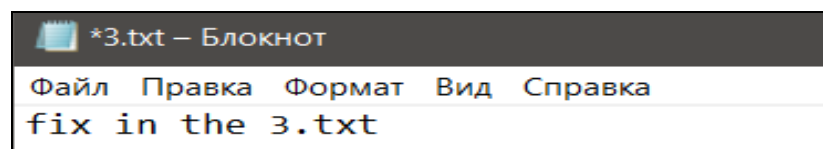


Рисунок 12. Добавил изменения в файл “2.txt”

```
commit 6518a6ac31f48e31ce98347f4631da3151e39f46 (HEAD -> branch_1)
Author: kulesh00v <oleg.kuleshov.1261@mai.ru>
Date: Mon Oct 23 20:34:53 2023 +0300

15 punkt
```

Рисунок 12. Добавил в индекс файлы “1.txt” и “2.txt”, произвёл коммит

```
(venv) PS C:\Users\User\PycharmProjects\Python_laba_3> git add 1.txt
(venv) PS C:\Users\User\PycharmProjects\Python_laba_3> git add 3.txt
(venv) PS C:\Users\User\PycharmProjects\Python_laba_3> git commit -m '16 pynkt'
[branch_2 da69b7c] 16 pynkt
 2 files changed, 2 insertions(+), 1 deletion(-)
(venv) PS C:\Users\User\PycharmProjects\Python_laba_3>
```

Рисунок 16. Перешёл на ветку “branch_2”, редактировал файлы “1.txt” и “3.txt”, добавил изменения в индекс и произвёл коммит

```
(venv) PS C:\Users\User\PycharmProjects\Python_laba_3> git checkout branch_1
Switched to branch 'branch_1'
(venv) PS C:\Users\User\PycharmProjects\Python_laba_3> 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 17. Слил изменения ветки “branch_2” в ветку “branch_1”, получив два вполне ожидаемых конфликта

```
(venv) PS C:\Users\User\PycharmProjects\Python_laba_3> git status
On branch branch_1
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
    both modified:   1.txt
    both modified:   3.txt
```

Рисунок 18. Команда “git status”, отображающая конфликты

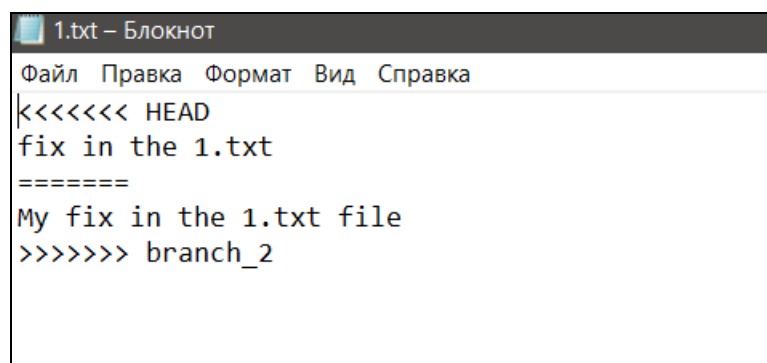


Рисунок 19. Конфликт в файле “1.txt”

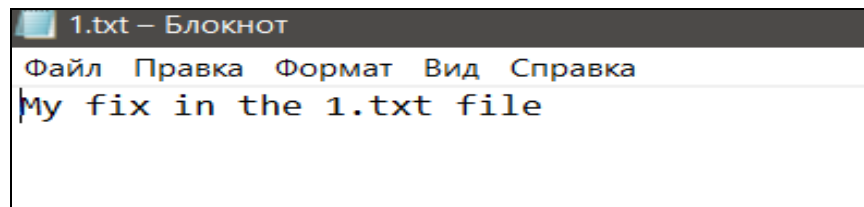


Рисунок 20. Решение первого конфликта вручную

```
(venv) PS C:\Users\User\PycharmProjects\Python_laba_3> git status
On branch branch_1
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Changes to be committed:
  modified:   1.txt

Unmerged paths:
  (use "git add <file>..." to mark resolution)
  both modified:   3.txt
```

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

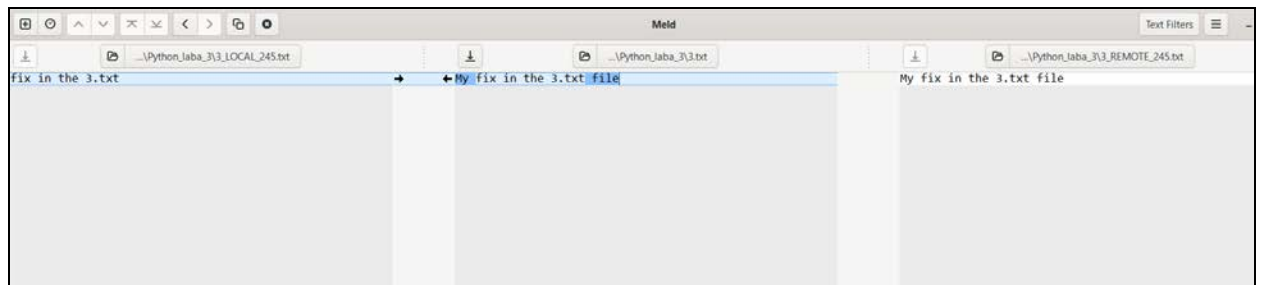


Рисунок 22. Решение второго конфликта с помощью утилиты “Meld”

```
Changes to be committed:
  modified:   1.txt
  modified:   3.txt
```

Рисунок 23. Два успешно урегулированных конфликта

```
(venv) PS C:\Users\User\PycharmProjects\Python_laba_3> git commit -m 'призвёл решение конфликтов'
[branch_1 7ba7c56] призвёл решение конфликтов
```

Рисунок 24. Произвёл коммит после решения конфликтов

```
(venv) PS C:\Users\User\PycharmProjects\Python_laba_3> git push origin branch_1
Enumerating objects: 23, done.
Counting objects: 100% (23/23), done.
Delta compression using up to 8 threads
Compressing objects: 100% (14/14), done.
Writing objects: 100% (22/22), 1.88 KiB | 643.00 KiB/s, done.
Total 22 (delta 6), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (6/6), done.
remote:
remote: Create a pull request for 'branch_1' on GitHub by visiting:
remote:   https://github.com/ku1esh00v/Python\_laba\_3/pull/new/branch\_1
remote:
To https://github.com/ku1esh00v/Python\_laba\_3.git
* [new branch]      branch_1 -> branch_1
```

Рисунок 25. Запустил ветку branch_1

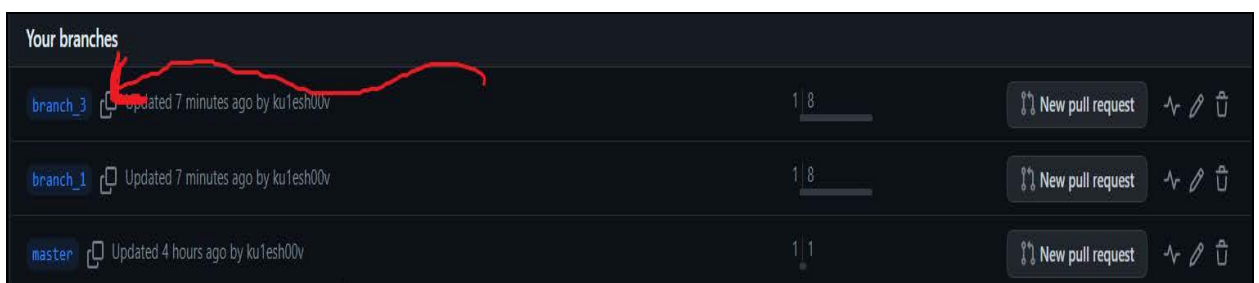


Рисунок 26. Создал ветку “branch_3” на удалённом репозитории

```
(venv) PS C:\Users\User\PycharmProjects\Python_laba_3> git checkout -b boba/branch_3
Switched to a new branch 'boba/branch_3'
```

Рисунок 27. Создал ветку отслеживания удалённой ветки “branch_3”

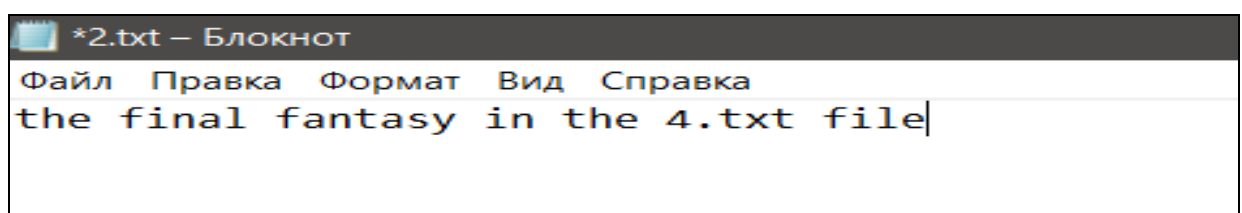


Рисунок 28. После перехода на ветку “branch_3” внёс изменения в файл “2.txt”

```
(venv) PS C:\Users\User\PycharmProjects\Python_laba_3> git add 2.txt
(venv) PS C:\Users\User\PycharmProjects\Python_laba_3> git commit -m 'изменил 2.txt в третьей ветке'
```

Рисунок 29. Добавил изменения в индекс и произвёл коммит

```
(venv) PS C:\Users\User\PycharmProjects\Python_laba_3> git checkout master
Already on 'master'
(venv) PS C:\Users\User\PycharmProjects\Python_laba_3> git rebase branch_2
Successfully rebased and updated refs/heads/master.
```

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 “вливать” 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, относительно 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 звездочкой

(*). 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 скопировано 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 отдельных 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 request'ов и многое другое, что делает работу с ветками в git более эффективной.