

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»**  
**Вариант \_\_\_\_**

Выполнил:  
Дудкин Константин Александрович  
2 курс, группа ИВТ-б-о-22-1,  
09.03.01 «Информатика и  
вычислительная техника»  
направление «Программное  
обеспечение средств вычислительной  
техники и автоматизированных  
систем»,  
очная форма обучения

---

(подпись)

Руководитель практики:  
Кандидат технических наук, доцент  
кафедры инфокоммуникаций, доцент  
Воронкин Роман Александрович

---

(подпись)

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

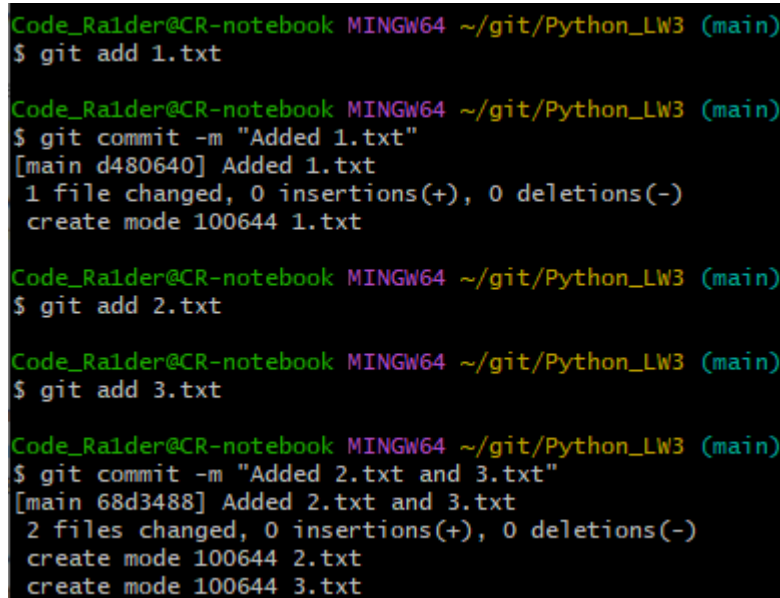
Ставрополь, 2023 г.

Тема: Основы ветвления Git

Цель: Исследование базовых возможностей по работе с локальными и удалёнными верками Git

### Порядок выполнения работы

1. Создал репозиторий, клонировал его на ПК и создал три текстовых файла - 1.txt, 2.txt, 3.txt
2. Сделал два отдельных коммита - один - с добавлением 1.txt, второй - с добавлением 2.txt и 3.txt



```
Code_Ra1der@CR-notebook MINGW64 ~/git/Python_LW3 (main)
$ git add 1.txt

Code_Ra1der@CR-notebook MINGW64 ~/git/Python_LW3 (main)
$ git commit -m "Added 1.txt"
[main d480640] Added 1.txt
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 1.txt

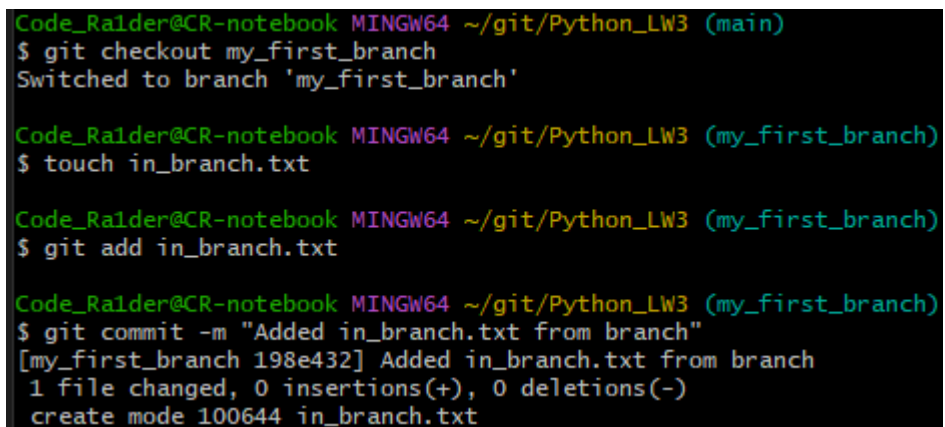
Code_Ra1der@CR-notebook MINGW64 ~/git/Python_LW3 (main)
$ git add 2.txt

Code_Ra1der@CR-notebook MINGW64 ~/git/Python_LW3 (main)
$ git add 3.txt

Code_Ra1der@CR-notebook MINGW64 ~/git/Python_LW3 (main)
$ git commit -m "Added 2.txt and 3.txt"
[main 68d3488] Added 2.txt and 3.txt
2 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 2.txt
create mode 100644 3.txt
```

Рисунок 1. Создание коммита для текстовых файлов

3. Создал ветку my\_first\_branch и, будучи в ней, создал in\_branch.txt



```
Code_Ra1der@CR-notebook MINGW64 ~/git/Python_LW3 (main)
$ git checkout my_first_branch
Switched to branch 'my_first_branch'

Code_Ra1der@CR-notebook MINGW64 ~/git/Python_LW3 (my_first_branch)
$ touch in_branch.txt

Code_Ra1der@CR-notebook MINGW64 ~/git/Python_LW3 (my_first_branch)
$ git add in_branch.txt

Code_Ra1der@CR-notebook MINGW64 ~/git/Python_LW3 (my_first_branch)
$ git commit -m "Added in_branch.txt from branch"
[my_first_branch 198e432] Added in_branch.txt from branch
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 in_branch.txt
```

Рисунок 2. Работа с файлом в ветке

4. Вернулся к ветке main, создал новую ветку new\_branch и, будучи в ней, изменил содержимое 1.txt, добавив строку с фразой

```
Code_Ra1der@CR-notebook MINGW64 ~/git/Python_LW3 (main)
$ git branch new_branch

Code_Ra1der@CR-notebook MINGW64 ~/git/Python_LW3 (main)
$ git checkout new_branch
Switched to branch 'new_branch'

Code_Ra1der@CR-notebook MINGW64 ~/git/Python_LW3 (new_branch)
$ echo "new row in 1.txt" >> 1.txt

Code_Ra1der@CR-notebook MINGW64 ~/git/Python_LW3 (new_branch)
$ git add 1.txt
warning: in the working copy of '1.txt', LF will be replaced by CRLF the next time Git touches it

Code_Ra1der@CR-notebook MINGW64 ~/git/Python_LW3 (new_branch)
$ git commit -m "Added row in 1.txt"
[new_branch f36a3b3] Added row in 1.txt
1 file changed, 1 insertion(+)
```

Рисунок 3. Создание новой ветки и работа с 1.txt

5. Вернулся в main и выполнил слияние main с my\_first\_branch и new\_branch

```
Code_Ra1der@CR-notebook MINGW64 ~/git/Python_LW3 (new_branch)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

Code_Ra1der@CR-notebook MINGW64 ~/git/Python_LW3 (main)
$ git merge -m "Merged first branch" my_first_branch
Updating 3891ac9..198e432
Fast-forward (no commit created; -m option ignored)
 in_branch.txt | 0
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 in_branch.txt

Code_Ra1der@CR-notebook MINGW64 ~/git/Python_LW3 (main)
$ git merge -m "Merged second branch" new_branch
Merge made by the 'ort' strategy.
1.txt | 1 +
1 file changed, 1 insertion(+)
```

Рисунок 4. Слияние main с my\_first\_branch и new\_branch

6. Удалил данные ветки и создал две новые

```

Code_Ra1der@CR-notebook MINGW64 ~/git/Python_LW3 (main)
$ git branch -d my_first_branch
Deleted branch my_first_branch (was 198e432).

Code_Ra1der@CR-notebook MINGW64 ~/git/Python_LW3 (main)
$ git branch -d new_branch
Deleted branch new_branch (was f36a3b3).

Code_Ra1der@CR-notebook MINGW64 ~/git/Python_LW3 (main)
$ git branch branch_1

Code_Ra1der@CR-notebook MINGW64 ~/git/Python_LW3 (main)
$ git branch branch_2

```

Рисунок 5. Удаление и создание новых веток

7. Перешёл на первую созданную ветку и выполнил изменения в файлах 1.txt и 3.txt. Закоммитил изменения

```

Code_Ra1der@CR-notebook MINGW64 ~/git/Python_LW3 (main)
$ git checkout branch_1
Switched to branch 'branch_1'

Code_Ra1der@CR-notebook MINGW64 ~/git/Python_LW3 (branch_1)
$ echo "Fix in 1.txt" > 1.txt

Code_Ra1der@CR-notebook MINGW64 ~/git/Python_LW3 (branch_1)
$ echo "Fix in 3.txt" > 3.txt

Code_Ra1der@CR-notebook MINGW64 ~/git/Python_LW3 (branch_1)
$ git add 1.txt 3.txt
warning: in the working copy of '1.txt', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of '3.txt', LF will be replaced by CRLF the next time Git touches it

Code_Ra1der@CR-notebook MINGW64 ~/git/Python_LW3 (branch_1)
$ git commit -m "Added fixes from branch_1"
[branch_1 3cccb44] Added fixes from branch_1
2 files changed, 2 insertions(+), 1 deletion(-)

```

Рисунок 6. Изменения в файлах в ветке branch\_1

8. Перешёл на вторую ветку и изменил содержимое файлов 1.txt и 3.txt. Закоммитил изменения

```

Code_Ra1der@CR-notebook MINGW64 ~/git/Python_LW3 (branch_1)
$ git checkout branch_2
Switched to branch 'branch_2'

Code_Ra1der@CR-notebook MINGW64 ~/git/Python_LW3 (branch_2)
$ echo "My fix in 1.txt" > 1.txt

Code_Ra1der@CR-notebook MINGW64 ~/git/Python_LW3 (branch_2)
$ echo "My fix in 3.txt" > 3.txt

Code_Ra1der@CR-notebook MINGW64 ~/git/Python_LW3 (branch_2)
$ git add 1.txt 3.txt
warning: in the working copy of '1.txt', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of '3.txt', LF will be replaced by CRLF the next time Git touches it

Code_Ra1der@CR-notebook MINGW64 ~/git/Python_LW3 (branch_2)
$ git commit -m "Added fix from branch_2"
[branch_2 c4b5c5e] Added fix from branch_2
2 files changed, 2 insertions(+), 1 deletion(-)

```

Рисунок 7. Изменения в файлах в ветке branch\_2

9. Выполнил слияние branch\_1 и branch\_2

```
Code_Ralder@CR-notebook MINGW64 ~/git/Python_LW3 (branch_1)
$ git merge -m "Merged branch_1 and branch_2" 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 8. Слияние...

К сожалению, во время слияния произошла ошибка

10. Исправил данный конфликт - 1.txt был изменён вручную, 3.txt был изменён через git mergetool (vimdiff)

11. Завершил слияние и отправил ветку на GitHub

```
Code_Ralder@CR-notebook MINGW64 ~/git/Python_LW3 (branch_1|MERGING)
$ git commit -m "Merged branch_1 and branch_2"
[branch_1 82b7ca4] Merged branch_1 and branch_2
```

Рисунок 9. Коммит слияния после решения конфликта

```
Code_Ralder@CR-notebook MINGW64 ~/git/Python_LW3 (branch_1)
$ git push --set-upstream origin branch_1
Enumerating objects: 22, done.
Counting objects: 100% (21/21), done.
Delta compression using up to 12 threads
Compressing objects: 100% (12/12), done.
Writing objects: 100% (19/19), 1.61 KiB | 1.61 MiB/s, done.
Total 19 (delta 8), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (8/8), completed with 1 local object.
remote:
remote: Create a pull request for 'branch_1' on GitHub by visiting:
remote:   https://github.com/CodeRalder/Python_LW3/pull/new/branch_1
remote:
To https://github.com/CodeRalder/Python_LW3.git
 * [new branch]      branch_1 -> branch_1
branch 'branch_1' set up to track 'origin/branch_1'.
```

Рисунок 10. Отправка ветки на GitHub

12. Средствами GitHub создал новую ветку в удалённом репозитории



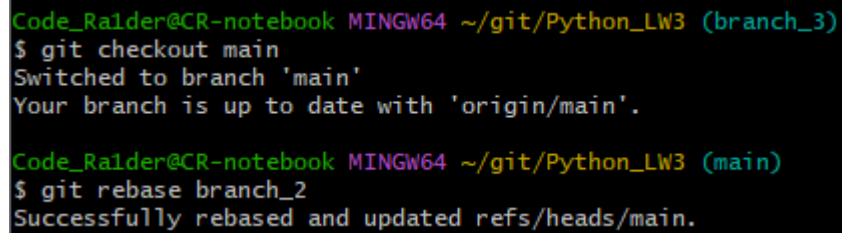
Рисунок 11. branch\_3 была создана на GitHub

13. Создал в локальном репозитории метку отслеживания новой ветки

```
Code_Ralder@CR-notebook MINGW64 ~/git/Python_LW3 (main)
$ git checkout --track origin/branch_3
Switched to a new branch 'branch_3'
branch 'branch_3' set up to track 'origin/branch_3'.
```

Рисунок 12. Создание метки

14. Из ветки branch\_3 изменил файл 2.txt и выполнил перемещение master на branch\_2



```
Code_Raider@CR-notebook MINGW64 ~/git/Python_LW3 (branch_3)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

Code_Raider@CR-notebook MINGW64 ~/git/Python_LW3 (main)
$ git rebase branch_2
Successfully rebased and updated refs/heads/main.
```

Рисунок 13. Перемещение master на branch\_2

15. Выполнил git push

### Ответы на вопросы

1. Ветка - последовательность коммитов репозитория, отклонившаяся от основной части в определённом моменте и идущая параллельно основному развитию репозитория

2. HEAD - указатель в репозитории, уникальный для каждой ветви, указывающий на последний коммит, выполненный данной ветвью.

Это означает, что если пользователь решит переключиться на другую ветвь репозитория, то указатель HEAD будет показывать вместо последнего коммита основной или сторонней ветки именно последний коммит той ветки, на которую пользователь решил переключиться

3. Существует три способа создания веток

- Команда `git branch` - создает ветку без переключения на неё
- Команда `git checkout -b` - создает ветку и сразу переключает пользователя на неё
- Создание веток в удалённых репозиториях - в GitHub есть возможность создать ветку относительно текущей ветки. Выполняется это через нажатие на название ветки выше списка файла

4. Чтобы узнать, в какой ветке сейчас находится пользователь, необходимо ввести команду `git branch` или `git status`

5. Для переключения между ветками используется команда `git checkout`

6. Удалённая ветка - ветка, располагающаяся на удалённом репозитории. Несмотря на то, что она не располагается на локальном репозитории, её можно скачать и продолжить работу с ней через ПК

7. Ветка отслеживания - локальная ветка, которая связана с удалённой веткой. Она автоматически фиксирует изменения, происходящие с удалённой веткой, что позволяет синхронизировать работу с ней

8. Для создания ветки отслеживания используется команда `git checkout`

9. Для этого применяют команду `git push origin <локальная_ветка>:<удалённая_ветка>`

10. `git fetch` и `git pull` отличаются тем, что `git pull` автоматически выполняет объединение изменений при установке удалённого репозитория в локальный, в то время как `git fetch` - нет. Однако функция у них общая - установка удалённого репозитория в локальный

11. Чтобы удалить локальную ветку, необходимо воспользоваться командой `git branch` с ключом `-d` или `--delete`

Чтобы удалить удалённую ветку необходимо воспользоваться командой `git push origin --delete`

12. Модель `git-flow` предполагает следующие основные типы веток:

- Main (Master) Branch - главная ветка, в которой располагается основная версия продукта

- Develop Branch - ветка разработки, в которой происходит основная разработка, а так же исправление функций и фичей

- Feature Branch(es) - ветка фичей, в которой создаются новые функции. В основном они дорабатываются в ветке Develop, а когда считаются готовыми - сливаются с Develop

- Release Branch(es) - ветка релизов, в которой фиксируется новая версия продукта. В ней происходит финальное тестирование, а затем слияние с Develop и Main

- Hotfix Branch(es) - ветка исправлений, как ясно из названия, выполняет функцию исправления критических ошибок продукта. Так же сливаются с Develop и Main при готовности

Работа с ветками git-flow осуществляется так:

- 1) Фичи создаются от Develop
- 2) Релизные ветки создаются перед выпуском новой версии. После тестирования сливаются с Main и Develop
- 3) Хотфиксы создаются от Main. При готовности сливаются с ней и с Develop

git-flow имеет ряд недостатков:

- 1) Модель git-flow сложна в реализации, она рассчитана на крупные проекты. Если дело касается простых проектов, она лишь замедлит разработку
- 2) Разработка с данной моделью может замедлиться из-за применения нескольких строго определённых веток. Особенно это касается веток фичей, релизов и хотфиксов
- 3) Ветки релизов могут стать сложными и ресурсозатратными, если у продукта имеется слишком много изменений
- 4) Данная модель не всегда может быть уместна для некоторых проектов разработки и может потребовать адаптации к конкретным потребностям

13. В GitHub Desktop есть возможность создания и управления ветками. Технически, данная программа является локальной копией GitHub, соответственно почти все функции, связанные с созданием веток в локальном репозитории, здесь присутствуют

Вывод: я изучил основные методы ветвления в Git, научился создавать ветки и управлять ими, а также рассмотрел возможности GitHub в управлении ветками