

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:
Репкин Александр Павлович
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., канд. техн. наук,
доцент кафедры инфокоммуникаций

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____
Ставрополь, 2023 г.

Тема: Основы ветвления Git.

Цель: исследовать базовые возможности по работе с локальными и удалёнными ветками Git.

Порядок выполнения работы:

1. В локальном репозитории создано 3 файла – 1.txt, 2.txt, 3.txt.

Этот компьютер > Локальный диск (C:) > Пользователи > yabuz > Python3 > Файлы и Программы

Имя	Дата изменения	Тип	Размер
1.txt	07.10.2023 17:38	Текстовый докум...	0 КБ
2.txt	07.10.2023 17:38	Текстовый докум...	0 КБ
3.txt	07.10.2023 17:38	Текстовый докум...	0 КБ

Рисунок 1. Три новых файла.

2. Инициализирован файл 1.txt и сделан коммит с сообщением “add 1.txt file”.

```
C:\Users\yabuz\Python3>git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   "\320\244\320\260\320\271\320\273\321\213 \320\270 \320\237\321\200\320\276\320\263\321\200\320\260\320\274\320\274\321\213\1.txt"

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    "\320\244\320\260\320\271\320\273\321\213 \320\270 \320\237\321\200\320\276\320\263\321\200\320\260\320\274\320\274\321\213\2.txt"
    "\320\244\320\260\320\271\320\273\321\213 \320\270 \320\237\321\200\320\276\320\263\321\200\320\260\320\274\320\274\321\213\3.txt"

C:\Users\yabuz\Python3>git commit -m "add 1.txt file"
[main ad80b99] add 1.txt file
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 "\320\244\320\260\320\271\320\273\321\213 \320\270 \320\237\321\200\320\276\320\263\321\200\320\260\320\274\320\274\321\213\1.txt"
```

Рисунок 2. Коммит с первым файлом.

3. Проиндексированы два оставшихся файла и перезаписан первый КОММИТ.

```
C:\Users\yabuz\Python3\Файлы и Программы>git add 2.txt 3.txt

C:\Users\yabuz\Python3\Файлы и Программы>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
```

Рисунок 3. Инициализация файлов 2.txt и 3.txt.

```
C:\Users\yabuz\Python3\Файлы и Программы>git commit --amend -m "add 2.txt and 3.txt."
[main 8c25b60] add 2.txt and 3.txt.
Date: Thu Oct 19 07:59:08 2023 +0300
3 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 "\320\244\320\260\320\271\320\273\321\213 \320\270 \320\237\321\200\320\276\320\263\321\200\320\260\320\274\320\274\321\213\1.txt"
create mode 100644 "\320\244\320\260\320\271\320\273\321\213 \320\270 \320\237\321\200\320\276\320\263\321\200\320\260\320\274\320\274\321\213\2.txt"
create mode 100644 "\320\244\320\260\320\271\320\273\321\213 \320\270 \320\237\321\200\320\276\320\263\321\200\320\260\320\274\320\274\321\213\3.txt"
```

Рисунок 4. Перезапись коммита.

4. Создана новая ветка – `my_first_branch`. Произведён переход на эту ветку. В ней же создан новый файл – `in_branch.txt`.

```
C:\Users\yabuz\Python3\Файлы и Программы>git branch "my_first_branch"

C:\Users\yabuz\Python3\Файлы и Программы>git branch
* main
  my_first_branch
```

Рисунок 5. Создание новой ветки.

```
C:\Users\yabuz\Python3\Файлы и Программы>git checkout "my_first_branch"
Switched to branch 'my_first_branch'

C:\Users\yabuz\Python3\Файлы и Программы>git branch
  main
* my_first_branch
```

Рисунок 6. Переход на новую ветку.

```
C:\Users\yabuz\Python3\Файлы и Программы>git status
On branch my_first_branch
Untracked files:
  (use "git add <file>..." to include in what will be committed)
  in_branch.txt

nothing added to commit but untracked files present (use "git add" to track)

C:\Users\yabuz\Python3\Файлы и Программы>git add in_branch.txt

C:\Users\yabuz\Python3\Файлы и Программы>git commit -m "First file in a first branch."
[my_first_branch 477d76c] First file in a first branch.
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 "\320\244\320\260\320\271\320\273\321\213 \320\270 \320\237\321\200\320\276\320\263\321\200\320\260\320\274\320\274\321\213\in_branch.txt"
```

Рисунок 7. Коммит нового файла в ветке.

5. Произведён переход на ветку `main`. Затем – создана новая ветка и сразу произведено переключение на неё.

```
C:\Users\yabuz\Python3\Файлы и Программы>git branch
  main
* my_first_branch

C:\Users\yabuz\Python3\Файлы и Программы>git checkout main
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

C:\Users\yabuz\Python3\Файлы и Программы>git branch
* main
  my_first_branch
```

Рисунок 8. Переход на ветку `main`.

```
C:\Users\yabuz\Python3\Файлы и Программы>git checkout -b new_branch
Switched to a new branch 'new_branch'

C:\Users\yabuz\Python3\Файлы и Программы>git branch
  main
  my_first_branch
* new_branch
```

Рисунок 9. Ветка new_branch.

6. Обновлён файл 1.txt, после чего был сразу произведён коммит. После перехода на ветку main произведено слияние веток main и my_first_branch при помощи команды git merge my_first_branch. Аналогично произведено слияние с new_branch. После этого новые ветки были удалены.

```
C:\Users\yabuz\Python3\Файлы и Программы>git status
On branch new_branch
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   1.txt

C:\Users\yabuz\Python3\Файлы и Программы>git commit -m "new row in the 1.txt file."
[new_branch 6805d48] new row in the 1.txt file.
1 file changed, 1 insertion(+)
```

Рисунок 10. Обновлён файл 1.txt.

```
C:\Users\yabuz\Python3\Файлы и Программы>git checkout main
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 1 commit.
(use "git push" to publish your local commits)

C:\Users\yabuz\Python3\Файлы и Программы>git merge my_first_branch
Updating 8c25b60..477d76c
Fast-forward
 .../in_branch.txt" | 0
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 ".../in_branch.txt"
320\274\320\274\321\213/in_branch.txt"
```

Рисунок 11. Слияние веток main и my_first_branch.

```
C:\Users\yabuz\Python3\Файлы и Программы>git merge new_branch
Merge made by the 'ort' strategy.
 .../1.txt" | 1 +
1 file changed, 1 insertion(+)
```

```
C:\Users\yabuz\Python3\Файлы и Программы>git branch
* main
  my_first_branch
  new_branch
```

Рисунок 12. Слияние веток main и new_branch.

```
C:\Users\yabuz\Python3\Файлы и Программы>git branch --delete my_first_branch
Deleted branch my_first_branch (was 477d76c).

C:\Users\yabuz\Python3\Файлы и Программы>git branch --delete new_branch
Deleted branch new_branch (was 6805d48).

C:\Users\yabuz\Python3\Файлы и Программы>git branch
* main
```

Рисунок 13. Удаление созданных двух веток.

7. Созданы две новые ветки – branch_1 и branch_2. Произведён переход на ветку branch_1, после чего произведены изменения в файлах 1.txt и 3.txt. Аналогично произведён переход в branch_2 и внесены изменения в файлы 1.txt и 3.txt. После этого была произведена попытка слияния веток branch_1 и branch_2, вызвавшая ошибку.

```
C:\Users\yabuz\Python3\Файлы и Программы>git checkout -b branch_1
Switched to a new branch 'branch_1'

C:\Users\yabuz\Python3\Файлы и Программы>git checkout -b branch_2
Switched to a new branch 'branch_2'

C:\Users\yabuz\Python3\Файлы и Программы>git checkout branch_1
Switched to branch 'branch_1'
```

Рисунок 14. Созданы ветки branch_1 и branch_2.

```
C:\Users\yabuz\Python3\Файлы и Программы>git status
On branch branch_1
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   1.txt
        modified:   3.txt

C:\Users\yabuz\Python3\Файлы и Программы>git commit -m "Changes in 1.txt and 3.txt files!"
[branch_1 8d05e62] Changes in 1.txt and 3.txt files!
2 files changed, 2 insertions(+), 1 deletion(-)
```

Рисунок 15. Произведены изменения в файлах 1.txt, 3.txt в branch_1.

```
C:\Users\yabuz\Python3\Файлы и Программы>git checkout branch_2
Switched to branch 'branch_2'

C:\Users\yabuz\Python3\Файлы и Программы>git branch
  branch_1
* branch_2
  main

C:\Users\yabuz\Python3\Файлы и Программы>git add 1.txt 3.txt

C:\Users\yabuz\Python3\Файлы и Программы>git commit -m "Changes in 1.txt and 3.txt files in branch_2!"
[branch_2 007be63] Changes in 1.txt and 3.txt files in branch_2!
2 files changed, 2 insertions(+), 1 deletion(-)
```

Рисунок 16. Произведены изменения в файлах 1.txt, 3.txt в branch_2.

```
C:\Users\yabuz\Python3\Файлы и Программы>git branch
* branch_1
  branch_2
  main

C:\Users\yabuz\Python3\Файлы и Программы>git merge branch_2
Auto-merging Файлы и Программы/1.txt
CONFLICT (content): Merge conflict in Файлы и Программы/1.txt
Auto-merging Файлы и Программы/3.txt
CONFLICT (content): Merge conflict in Файлы и Программы/3.txt
Automatic merge failed; fix conflicts and then commit the result.
```

Рисунок 17. Попытка слияния веток branch_1 и branch_2.

8. Произведена попытка исправить полученную ошибку при слиянии. Файл 1.txt был дополнительно изменён вручную, придавая ему наиболее желаемый вид. Файл 3.txt был исправлен при помощи git mergetool.

```
1.txt - Блокнот
Файл Правка Формат Вид Справка
|<<<<<< HEAD
fix in the 1.txt
=====
My fix in the 1.txt
>>>>>> branch_2
```

Рисунок 18. Файл 1.txt до изменений.




Рисунок 19. Файл 1.txt после изменений.

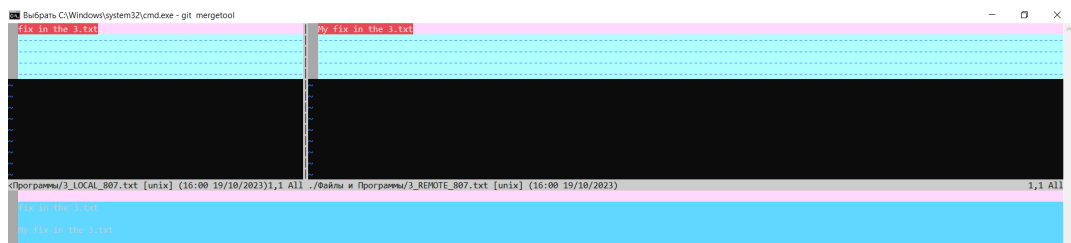


Рисунок 20. Исправление конфликта при помощи git mergetool.

9. Ветка branch 1 отправлена в удалённый репозиторий. .

```
C:\Users\yabuz\Python3>git push origin branch_1
Enumerating objects: 20, done.
Counting objects: 100% (20/20), done.
Delta compression using up to 16 threads
Compressing objects: 100% (15/15), done.
Writing objects: 100% (19/19), 1.49 KiB | 305.00 KiB/s, done.
Total 19 (delta 8), reused 0 (delta 0), pack-reused 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/AlexRepkin/Python3/pull/new/branch_1
remote:
To https://github.com/AlexRepkin/Python3.git
 * [new branch]      branch 1 -> branch 1
```

Рисунок 21. Отправка ветки branch 1 на сервер.

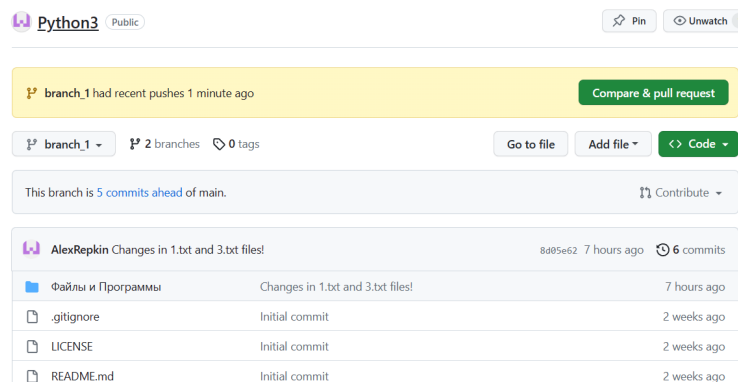


Рисунок 22. Отправленная ветка branch 1.

10. Создана удалённая ветка `branch_3`. Локально создана ветка отслеживания ветки `branch_3`.

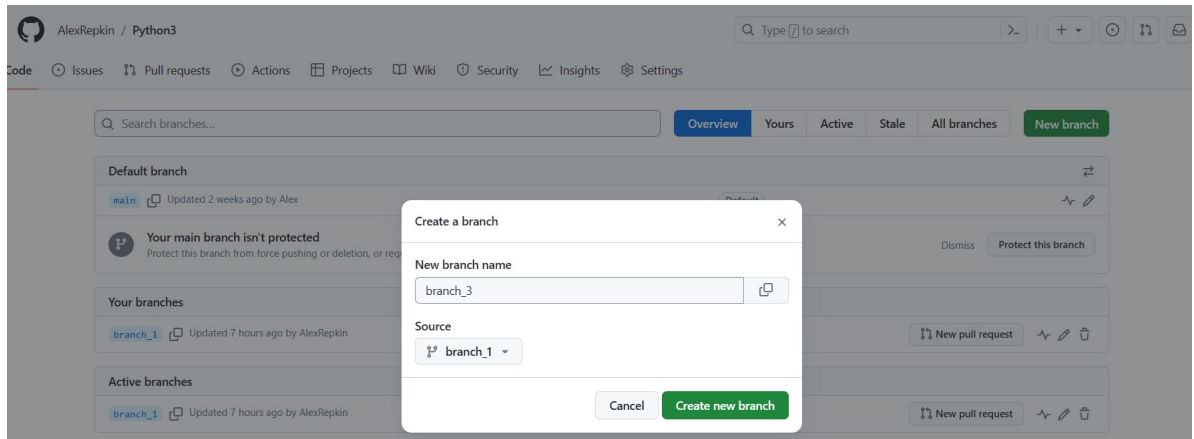


Рисунок 23. Создание ветки `branch_3`.

```
C:\Users\yabuz\Python3\Файлы и Программы>git checkout -t origin/branch_3
Switched to a new branch 'branch_3'
M       "\320\244\320\260\320\271\320\273\321\213 \320\270 \320\237\321\200\320\276\320\263\321\200\320\260\320\274\320\274\321\213\1.txt"
M       "\320\244\320\260\320\271\320\273\321\213 \320\270 \320\237\321\200\320\276\320\263\321\200\320\260\320\274\320\274\321\213\3.txt"
branch 'branch_3' set up to track 'origin/branch_3'.
```

Рисунок 24. Создана ветка отслеживания `branch_3`

11. В ветке `branch_3` изменено содержание файла `2.txt`.

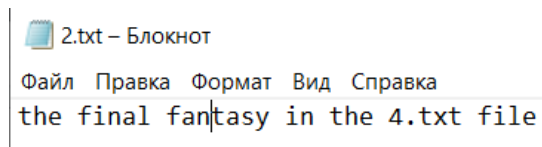


Рисунок 25. Файл `2.txt` в ветке `branch_3`.

```
C:\Users\yabuz\Python2\Программы\Python2_Tasks_On_C++\Python2_Tasks_On_C++>git reset --hard HEAD~1
HEAD is now at b74e269 No symbols abilities here yet
```

Рисунок 26. Команда `git reset --hard HEAD~1`.

12. Ветки отправлены на сервер.

```
C:\Users\yabuz\Python3>git push origin branch_2
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (5/5), 436 bytes | 109.00 KiB/s, done.
Total 5 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'branch_2' on GitHub by visiting:
remote:   https://github.com/AlexRepkin/Python3/pull/new/branch_2
remote:
To https://github.com/AlexRepkin/Python3.git
 * [new branch]      branch_2 -> branch_2
```

Рисунок 27. Отправка ветки `branch_2` на сервер.

13. Ответы на вопросы.

1) Что такое ветка?

Ответ: ветка — это подвижный указатель на определенный коммит в истории проекта. Ветки позволяют разработчикам работать параллельно, создавать новые функции и исправлять ошибки, не затрагивая основной код.

2) Что такое HEAD?

Ответ: HEAD – указатель, ссылающийся на определённый коммит в репозитории. Данный коммит будет являться родителем для следующего созданного коммита.

3) Способы создания веток?

Ответ: для создания новых веток существует команда `git branch` “название”, только создающая, но не переключающая на неё. Также, существует команда `git checkout -b` “название” “хеш-сумма коммита или название тега”, если необходимо создать новую ветку на основе определённого тега\коммита и сразу переключиться на неё.

4) Как узнать текущую ветку?

Ответ: для проверки текущей ветки существует команда `git branch`. Также, можно узнать название ветки, если использовать команду `git log –decorate` (Но тогда выведутся все ветки, указывающие на данный коммит).

5) Как переключаться между ветками?

Ответ: для переключения между ветками существует команда `git checkout` “название ветки”.

6) Что такое удалённая ветка?

Ответ: удалённая ветка – копия локальной ветки, хранимая в удалённом репозитории. Получить информацию об удалённом репозитории можно с помощью команды `git remote show` “название” (или `git ls-remote` “название”).

7) Что такое ветка отслеживания?

Ответ: ветка отслеживания - ссылка на определённое состояние удалённой ветки, это локальные ветки, которые нельзя перемещать. Git перемещает их автоматически при любой коммуникации с удалённым

репозиторием, чтобы гарантировать точное соответствие с ним. Имена веток отслеживания имеют вид “название сервера”/”название ветки”.

8) Как создать ветку отслеживания?

Ответ: для создания веток отслеживания создано дополнение `–track` (или `-t`) для `git checkout`. Получаемая команда – `git checkout -t “название сервера”/ “название ветки”`.

9) Как отправить изменения из локальной ветки в удалённую?

Ответ: для отправки изменений из локальной ветки в удалённую существует команда `git push “удалённый сервер” “название ветки”`. При необходимости, можно изменить название удалённой ветки, добавив новое название к названию ветки через `“:”`.

10) В чём отличие команд `git fetch` и `git pull`?

Ответ: `git fetch` - получает с сервера все отсутствующие изменения, не изменяя при этом состояние локальной директории. `git pull` - команда `git fetch`, за которой непосредственно следует команда `git merge`, объединяющая локальную и удалённую версии репозитория.

11) Как удалить локальную и удалённые ветки?

Ответ: для удаления ветки на удалённом сервере существует команда `git push “удалённый сервер” –delete(или просто -d) “название ветки”`. Однако данная команда лишь удаляет указатель на сервере. Если нужно удалить локальную ветку, то можно использовать команду `git branch -d “название ветки”`.

12) Какие основные типы веток присутствуют в модели `git-flow`? Как организована работа с ветками в модели `git-flow`? В чём недостатки `git-flow`?

Ответ: `Git-flow` — альтернативная модель ветвления `Git`, в которой используются функциональные и несколько основных веток. В `Git-flow` используются следующие типы веток:

Функциональные ветки - создаются на основе последней ветки разработки. Когда работа с функцией завершена, ветка сливается с `develop`-веткой. Создаётся при помощи `git flow feature start “название ветки”` (если есть

библиотека) или `git checkout -b "название ветки"` от ветки разработки. Если необходимо закончить работу с веткой, то при наличии библиотеки можно воспользоваться командой `git flow feature finish "название ветки"` или командами `git checkout "develop"` и `git merge "название ветки"`;

Ветки выпуска - когда в ветке `develop` оказывается достаточно функций для выпуска, от ветки разработки создается ветка выпуска. Создание этой ветки запускает следующий цикл релиза, и с этого момента новые функции добавить больше нельзя — допускается лишь исправление ошибок, создание документации и решение других задач, связанных с релизом. Когда подготовка к поставке завершается, ветка `release` сливается с `main` и ей присваивается номер версии. Кроме того, нужно выполнить ее слияние с веткой `develop`, в которой с момента создания ветки релиза могли возникнуть изменения. Для создания такой ветки можно воспользоваться командами `git checkout "develop"` и `git checkout -b release/"версия"` или, если есть библиотека, можно воспользоваться командой `git flow release start "версия"`. Для завершения работы можно использовать команды `git checkout main` и `git merge release/"версия"`, или, если есть библиотека, можно воспользоваться `git flow release finish "версия"`;

Ветки исправления - используются для быстрого внесения исправлений в рабочие релизы. Они создаются на основе `main`, а не `develop`. Это единственная ветка, которую нужно обязательно создавать напрямую от `main`. Как только исправление завершено, эту ветку следует слить с `main` и `develop` (или текущей веткой `release`), а ветке `main` присвоить обновленный номер версии. Для создания такой ветки можно воспользоваться командами `git checkout main` и `git checkout -b "название ветки"` или, если есть библиотека, можно воспользоваться командой `git flow hotfix start "название ветки"`. Для завершения работы можно использовать команды `git checkout main`, `git merge "название ветки"`, `git checkout develop`, `git merge "название ветки"`, `git branch -D "название ветки"`, или, если есть библиотека, можно воспользоваться `git flow hotfix finish "название ветки"`;

Также присутствуют ветка разработки (develop, в ней хранится полная история проекта. Её требуется создать самостоятельно и отправить на сервер вручную, или же можно использовать библиотеку git-flow и воспользоваться командой git flow init) и главная ветка (main, в ней хранится сокращённая история проекта).

Минусы Git-flow- Такие долгосрочные функциональные ветки требуют тесного взаимодействия разработчиков при слиянии и создают повышенный риск отклонения от магистральной ветки. В них также могут присутствовать конфликтующие обновления.

13) Какие инструменты для работы с ветками присутствуют в BitBucket?

Ответ: BitBucket предоставляет возможность использования всех инструментов, приведённых в данной лабораторной работе. Ветки можно сливать, удалять, создавать, обновлять идентичными командами.

Вывод: в ходе выполнения лабораторной работы были исследованы базовые возможности работы с локальными и удалёнными ветками Git.