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»

Выполнила:
Беседина Инга Олеговна
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и вычислительная техника (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р. А., канд. технических 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 удаленными ветками Git

Порядок выполнения работы:

```
besed@DESKTOP-FUPA64L MINGW64 /c/Rep/Python_PR_1_3 (main)
$ git add 1.txt

besed@DESKTOP-FUPA64L MINGW64 /c/Rep/Python_PR_1_3 (main)
$ git commit -m "add 1.txt file"
[main 0166769] add 1.txt file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 1.txt
```

Рисунок 1. Индексация и коммит первого файла

```
besed@DESKTOP-FUPA64L MINGW64 /c/Rep/Python_PR_1_3 (main)
$ git add 2.txt

besed@DESKTOP-FUPA64L MINGW64 /c/Rep/Python_PR_1_3 (main)
$ git add 3.txt
```

Рисунок 2. Индексация второго и третьего фала

```
besed@DESKTOP-FUPA64L MINGW64 /c/Rep/Python_PR_1_3 (main)
$ git commit --amend -m "add 2.txt and 3.txt"
[main de59a37] add 2.txt and 3.txt
 Date: Thu Sep 21 14:05:37 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 3. Перезапись коммита

```
besed@DESKTOP-FUPA64L MINGW64 /c/Rep/Python_PR_1_3 (main)
$ git branch my_first_branch

besed@DESKTOP-FUPA64L MINGW64 /c/Rep/Python_PR_1_3 (main)
$ git checkout my_first_branch
Switched to branch 'my_first_branch'
```

Рисунок 4. Создание новой ветки

```
besed@DESKTOP-FUPA64L MINGW64 /c/Rep/Python_PR_1_3 (my_first_branch)
$ git add in_branch.txt

besed@DESKTOP-FUPA64L MINGW64 /c/Rep/Python_PR_1_3 (my_first_branch)
$ git commit -m "add in_branch.txt"
[my_first_branch f70c5fb] add in_branch.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 in_branch.txt
```

Рисунок 5. Создание нового файла in_branch.txt, коммит изменений

```
besed@DESKTOP-FUPA64L MINGW64 /c/Rep/Python_PR_1_3 (main)
$ git branch new_branch

besed@DESKTOP-FUPA64L MINGW64 /c/Rep/Python_PR_1_3 (main)
$ git checkout new_branch
Switched to branch 'new_branch'

besed@DESKTOP-FUPA64L MINGW64 /c/Rep/Python_PR_1_3 (new_branch)
$ git add 1.txt

besed@DESKTOP-FUPA64L MINGW64 /c/Rep/Python_PR_1_3 (new_branch)
$ git commit -m "new row in the 1.txt file"
[new_branch c3fb449] new row in the 1.txt file
 1 file changed, 1 insertion(+)
```

Рисунок 6. Создание новой ветки

```
besed@DESKTOP-FUPA64L MINGW64 /c/Rep/Python_PR_1_3 (new_branch)
$ git add 1.txt

besed@DESKTOP-FUPA64L MINGW64 /c/Rep/Python_PR_1_3 (new_branch)
$ git commit -m "new row in the 1.txt file"
[new_branch c3fb449] new row in the 1.txt file
 1 file changed, 1 insertion(+)
```

Рисунок 7. Коммит изменений

```
besed@DESKTOP-FUPA64L MINGW64 /c/Rep/Python_PR_1_3 (main)
$ git merge my_first_branch
Updating de59a37..f70c5fb
Fast-forward
 in_branch.txt | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 in_branch.txt
```

Рисунок 8. Слияние веток main и my_first_branch

```
besed@DESKTOP-FUPA64L MINGW64 /c/Rep/Python_PR_1_3 (main)
$ git merge new_branch
Merge made by the 'ort' strategy.
 1.txt | 1 +
 1 file changed, 1 insertion(+)
```

Рисунок 9. Слияние веток main и new_branch

```
besed@DESKTOP-FUPA64L MINGW64 /c/Rep/Python_PR_1_3 (main)
$ git branch -d my_first_branch
Deleted branch my_first_branch (was f70c5fb).

besed@DESKTOP-FUPA64L MINGW64 /c/Rep/Python_PR_1_3 (main)
$ git branch -d new_branch
Deleted branch new_branch (was c3fb449).
```

Рисунок 10. Удаление веток new_branch и my_first_branch

```
besed@DESKTOP-FUPA64L MINGW64 /c/Rep/Python_PR_1_3 (main)
$ git branch branch_1

besed@DESKTOP-FUPA64L MINGW64 /c/Rep/Python_PR_1_3 (main)
$ git branch branch_2
```

Рисунок 11. Создание веток

```
besed@DESKTOP-FUPA64L MINGW64 /c/Rep/Python_PR_1_3 (main)
$ git checkout branch_1
Switched to branch 'branch_1'
```

Рисунок 12. Переход на ветку

```
besed@DESKTOP-FUPA64L MINGW64 /c/Rep/Python_PR_1_3 (branch_1)
$ git commit -m "edit files 1.txt 3.txt"
[branch_1 e64608f] edit files 1.txt 3.txt
 2 files changed, 2 insertions(+), 1 deletion(-)
```

Рисунок 13. Коммит изменений

```
besed@DESKTOP-FUPA64L MINGW64 /c/Rep/Python_PR_1_3 (branch_1)
$ git checkout branch_2
Switched to branch 'branch_2'
```

Рисунок 14. Переход на ветку

```
besed@DESKTOP-FUPA64L MINGW64 /c/Rep/Python_PR_1_3 (branch_2)
$ git commit -m "edit files 1.txt 3.txt on the branch_2"
[branch_2 6237dba] edit files 1.txt 3.txt on the branch_2
 2 files changed, 2 insertions(+), 1 deletion(-)
```

Рисунок 15. Коммит изменений

```
$ git merge branch_2
Auto-merging txt/1.txt
CONFLICT (content): Merge conflict in txt/1.txt
Auto-merging txt/3.txt
CONFLICT (content): Merge conflict in txt/3.txt
Automatic merge failed; fix conflicts and then commit the result.
```

Рисунок 16. Конфликт при слиянии веток

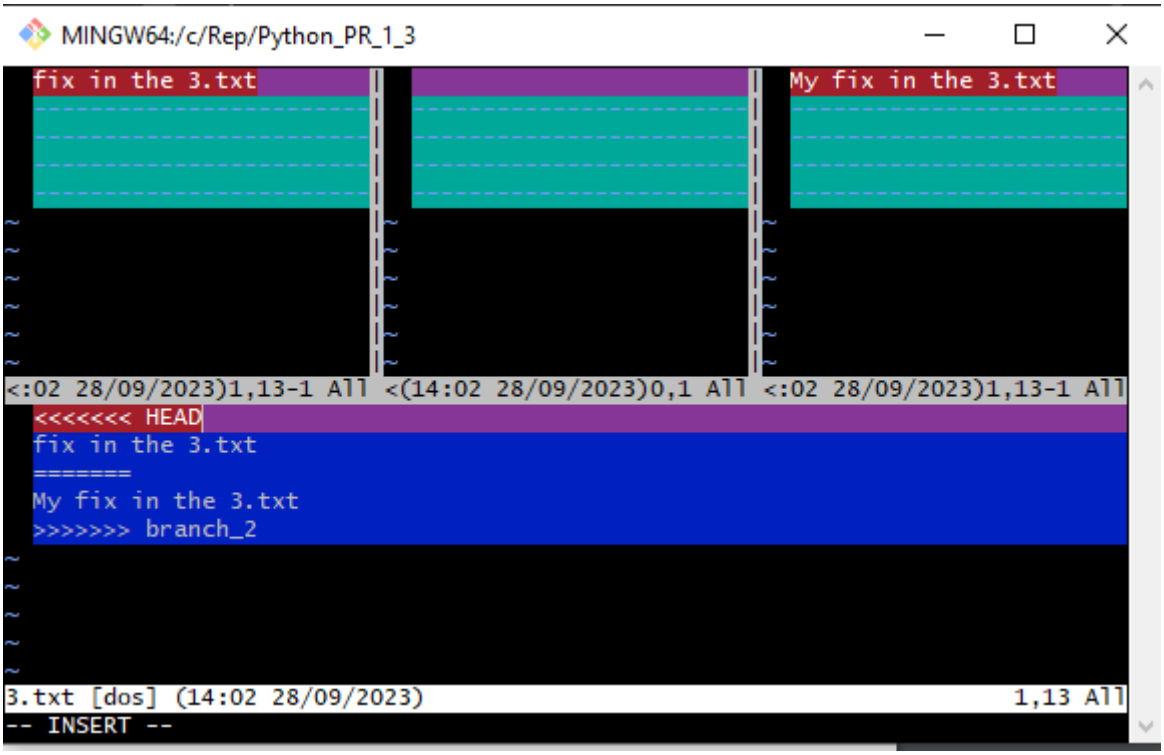

Рисунок 17. Git mergetool

```
besed@DESKTOP-FUPA64L MINGW64 /c/Rep/Python_PR_1_3 (branch_1|MERGING)
$ git commit -m "fix conflicts"
[branch_1 f90bc6e] fix conflicts
```

Рисунок 18. Коммит после разрешения конфликта

```
besed@DESKTOP-FUPA64L MINGW64 /c/Rep/Python_PR_1_3 (branch_1)
$ git push origin branch_1
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 280 bytes | 70.00 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/IngaBesedina/Python_PR_1_3.git
   f90bc6e..0ce7bd0  branch_1 -> branch_1
```

Рисунок 19. Отправка ветки на GitHub

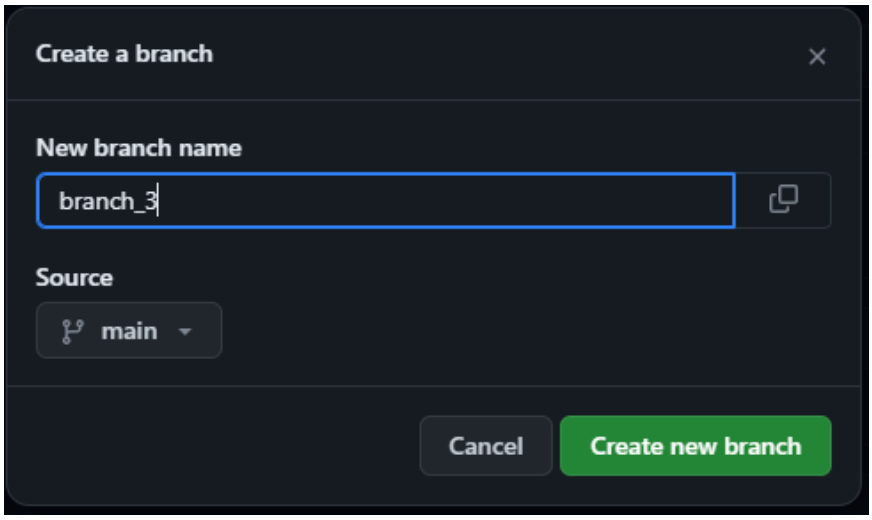

Рисунок 20. Создание удаленной ветки средствами GitHub

```
Admin@DESKTOP-FPU5JA3 MINGW64 /d/Rep/Python_LR_1_3 (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 21. Создание ветки отслеживания удалённой ветки

```
Admin@DESKTOP-FPU5JA3 MINGW64 /d/Rep/Python_LR_1_3 (branch_3)
$ git rebase main
Current branch branch_3 is up to date.
```

Рисунок 22. Перемещение ветки main на branch_3

```
Admin@DESKTOP-FPU5JA3 MINGW64 /d/Rep/Python_LR_1_3 (main)
$ git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 417 bytes | 417.00 KiB/s, done.
Total 4 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/IngaBesedina/Python_LR_1_3.git
   feb92af..3741286  main -> main
```

Рисунок 23. Отправка изменений на GitHub

Контрольные вопросы:

1. Ветка в Git это подвижный указатель на один из коммитов. Обычно ветка указывает на последний коммит в цепочке коммитов. Ветка берет свое начало от какого-то одного коммита

2. HEAD – это указатель, задача которого ссылаться на определенный коммит в репозитории. Во-первых, HEAD – это указатель на коммит в вашем репозитории, который станет родителем следующего коммита. Во-вторых, HEAD указывает на коммит, относительного которого будет создана рабочая копия во-время операции checkout

3. Способы создания веток: командой git branch <branch_name>, средствами GitHub

4. Узнать текущую ветку можно с помощью команды git log –oneline --decorate, которая покажет куда указывают указатели веток

5. Переключаться между ветками можно с помощью команды git checkout <branch_name>

6. Удалённые ветки — это ссылки на состояние веток в удалённых репозиториях.

7. Ветки слежения — это ссылки на определённое состояние удалённых веток. Это локальные ветки, которые нельзя перемещать; Git перемещает их автоматически при любой коммуникации с удаленным репозиторием, чтобы гарантировать точное соответствие с ним.

8. Создать ветку отслеживания можно с помощью команды git checkout –track <remote>/<branch>

9. Отправить изменения из локальной ветки в удаленную ветку git push <remote> <branch>

10. Команда git fetch загружает изменения из удаленного репозитория в локальный репозиторий, но не применяет их к текущей ветке. Команда git pull также загружает изменения из удаленного репозитория в локальный репозиторий, но автоматически объединяет эти изменения с текущей веткой.

11. Удалить локальную ветку git branch –d <branch_name>, удалить удалённую ветку git push <remote> --delete <branch_name>

12. Основные типы веток в модели git-flow: Master - основная ветка, которая содержит стабильный код и отображает текущую версию продукта. Develop - ветка разработки, которая содержит последние изменения и новые функции, но еще не готова для релиза. Feature - ветки, создаваемые для разработки новых функций или исправления ошибок. Каждая функция должна быть разработана в отдельной ветке и объединена с веткой Develop после завершения. Release - ветки, создаваемые для подготовки к выпуску новой версии продукта. В этой ветке происходит тестирование и исправление ошибок перед выпуском. Hotfix - ветки, создаваемые для быстрого исправления критических ошибок в текущей версии продукта.

В работе с ветками в модели git-flow используется строгий процесс, который помогает обеспечить стабильность и качество кода. Каждая задача начинается с создания отдельной ветки, которая основывается на соответствующей ветке (например, Feature - на Develop). После завершения работы над задачей, изменения объединяются с соответствующей веткой (например, Feature - с Develop) и удаляется ветка задачи.

Недостатки git-flow заключаются в том, что он может быть слишком сложным для небольших команд или проектов, где не требуется такой уровень контроля и структурированности. Также, использование модели git-flow может привести к большому количеству веток, что может затруднить работу с кодом и усложнить процесс обновления и слияния изменений

13. GitHub Desktop - это графический интерфейс для работы с Git, который предоставляет удобные инструменты для создания и управления ветками.

Создание ветки - пользователь может создать новую ветку из текущей ветки или из другой ветки. Для этого нужно нажать на кнопку "Create a new branch" и ввести название новой ветки.

Переключение между ветками - пользователь может переключаться между ветками, выбрав нужную ветку из выпадающего списка.

Объединение веток - пользователь может объединять ветки, выбрав нужную ветку и нажав на кнопку "Merge into current branch".

Удаление веток - пользователь может удалить ветку, выбрав нужную ветку и нажав на кнопку "Delete branch".

Отслеживание изменений - GitHub Desktop позволяет отслеживать изменения в ветках и сравнивать изменения между ветками. Работа с конфликтами - при объединении веток, GitHub Desktop показывает конфликты и предлагает решения для их разрешения.

Вывод: в ходе выполнения лабораторной работы было проведено исследование базовых возможностей по работе с локальными и удаленными ветками Git.