

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

Институт цифрового развития
Кафедра инфокоммуникаций

«Основы ветвления Git»

ОТЧЕТ
по лабораторной работе №3
дисциплины
«Основы программной инженерии»

Выполнил:

Гълбачева Доротея Андреева
2 курс, группа ПИЖ-б-о-21-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка
и сопровождение программного
обеспечения», очная форма обучения

(подпись)

Проверил:

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2022 г.

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

Методика и порядок выполнения работы:

1. Создать три файла: 1.txt, 2.txt, 3.txt.

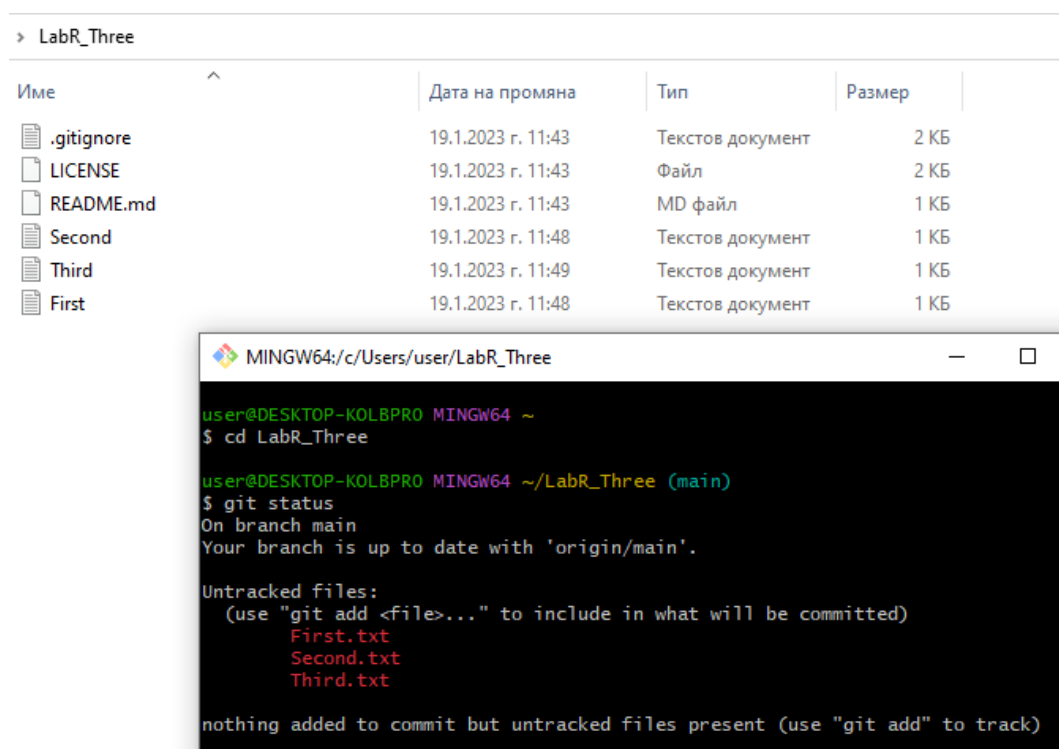


Рисунок 3.1 – Созданные файлы

2. Проиндексировать первый файл и сделать коммит с комментарием "add 1.txt file".

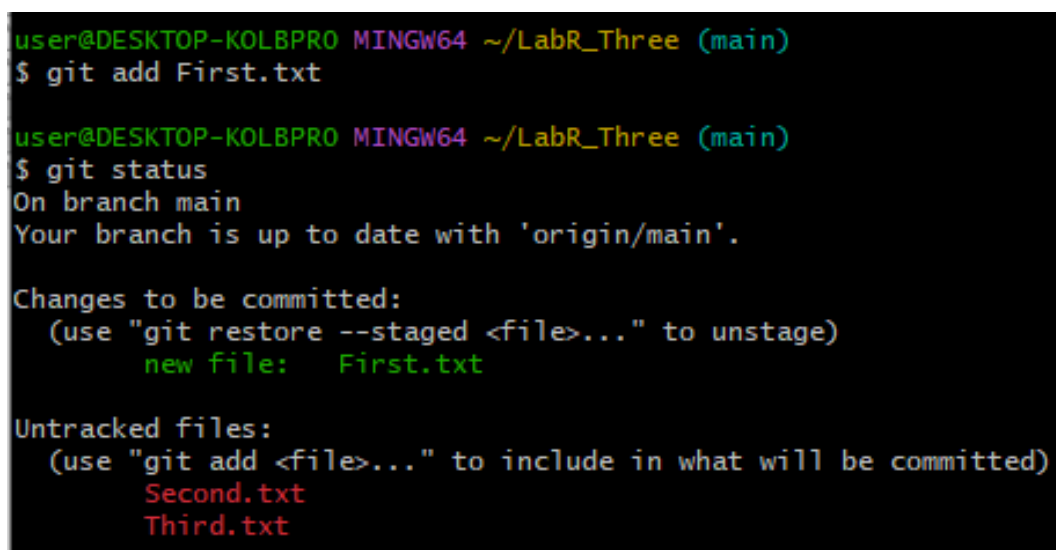


Рисунок 3.2 – Индексация первого файла

3. Проиндексировать второй и третий файлы.

```
user@DESKTOP-KOLBPRO MINGW64 ~/LabR_Three (main)
$ git add Second.txt Third.txt

user@DESKTOP-KOLBPRO MINGW64 ~/LabR_Three (main)
$ 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:   First.txt
    new file:   Second.txt
    new file:   Third.txt
```

Рисунок 3.3 – Индексация второго и третьего файла

4. Перезаписать уже сделанный коммит с новым комментарием "add 2.txt and 3.txt."

```
user@DESKTOP-KOLBPRO MINGW64 ~/LabR_Three (My_First_Branch)
$ git commit -m "add Second.txt and Third.txt"
[My_First_Branch d0704ce] add Second.txt and Third.txt
3 files changed, 3 insertions(+)
create mode 100644 First.txt
create mode 100644 Second.txt
create mode 100644 Third.txt
```

Рисунок 3.4 – Перезапис коммита

5. Создать новую ветку my_first_branch.

```
user@DESKTOP-KOLBPRO MINGW64 ~/LabR_Three (main)
$ git branch My_First_Branch

user@DESKTOP-KOLBPRO MINGW64 ~/LabR_Three (main)
$ git checkout My_First_Branch
Switched to branch 'My_First_Branch'
A       First.txt
A       Second.txt
A       Third.txt
```

Рисунок 3.4 – Создание новой ветки «My_First_Branch»

6. Перейти на ветку и создать новый файл in_branch.txt, закоммитить изменения.

```

user@DESKTOP-KOLBPRO MINGW64 ~/LabR_Three (My_First_Branch)
$ git add In_Branch.txt

user@DESKTOP-KOLBPRO MINGW64 ~/LabR_Three (My_First_Branch)
$ git commit -m "add In_Branch.txt"
[My_First_Branch fedc70e] add In_Branch.txt
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 In_Branch.txt

```

Рисунок 3.6 – Индексация и коммит файла

7. Вернуться на ветку master. Создать и сразу перейти на ветку new_branch. Сделать изменения в файле 1.txt, добавить строчку “new row in the 1.txt file”, закоммитить изменения.

```

user@DESKTOP-KOLBPRO MINGW64 ~/LabR_Three (main)
$ git checkout new_branch
Switched to branch 'new_branch'

user@DESKTOP-KOLBPRO MINGW64 ~/LabR_Three (new_branch)
$ git add First.txt

user@DESKTOP-KOLBPRO MINGW64 ~/LabR_Three (new_branch)
$ git commit -m "Change First.txt"
[new_branch e06eb19] Change First.txt
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 First.txt

```

Рисунок 3.7 – Возврат к ветке (main), создание новой ветки и сразу переходом на нее и добавление изменений в файл с последующим

КОММИТОМ

8. Перейти на ветку master и слить ветки master и my_first_branch, после чего слить ветки master и new_branch.

```

user@DESKTOP-KOLBPRO MINGW64 ~/LabR_Three (main)
$ git merge My_new_branch
Already up to date.

user@DESKTOP-KOLBPRO MINGW64 ~/LabR_Three (main)
$ git merge New_branch
Already up to date.

```

Рисунок 3.8 – Слияние веток

9. Удалить ветки my_first_branch и new_branch.

```
user@DESKTOP-KOLBPR0 MINGW64 ~/LabR_Three (main)
$ git branch -d My_new_branch
Deleted branch My_new_branch (was e06eb19).

user@DESKTOP-KOLBPR0 MINGW64 ~/LabR_Three (main)
$ git branch -d New__Branch
Deleted branch New__Branch (was e06eb19).
```

Рисунок 3.9 – Удаление веток

10. Создать ветки branch_1 и branch_2.

```
user@DESKTOP-KOLBPR0 MINGW64 ~/LabR_Three (main)
$ git branch branch_1

user@DESKTOP-KOLBPR0 MINGW64 ~/LabR_Three (main)
$ git branch branch_2
```

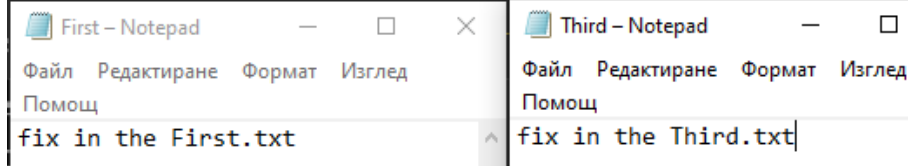
Рисунок 3.10 – Создание веток

11. Перейти на ветку branch_1 и изменить файл 1.txt, удалить все содержимое и добавить текст “fix in the 1.txt”, изменить файл 3.txt, удалить все содержимое и добавить текст “fix in the 3.txt”, закоммитить изменения.

```
user@DESKTOP-KOLBPR0 MINGW64 ~/LabR_Three (main)
$ git checkout branch_1
Switched to branch 'branch_1'
M       First.txt
A       In_Branch.txt
A       Second.txt
A       Third.txt

user@DESKTOP-KOLBPR0 MINGW64 ~/LabR_Three (branch_1)
$ git add First.txt Third.txt

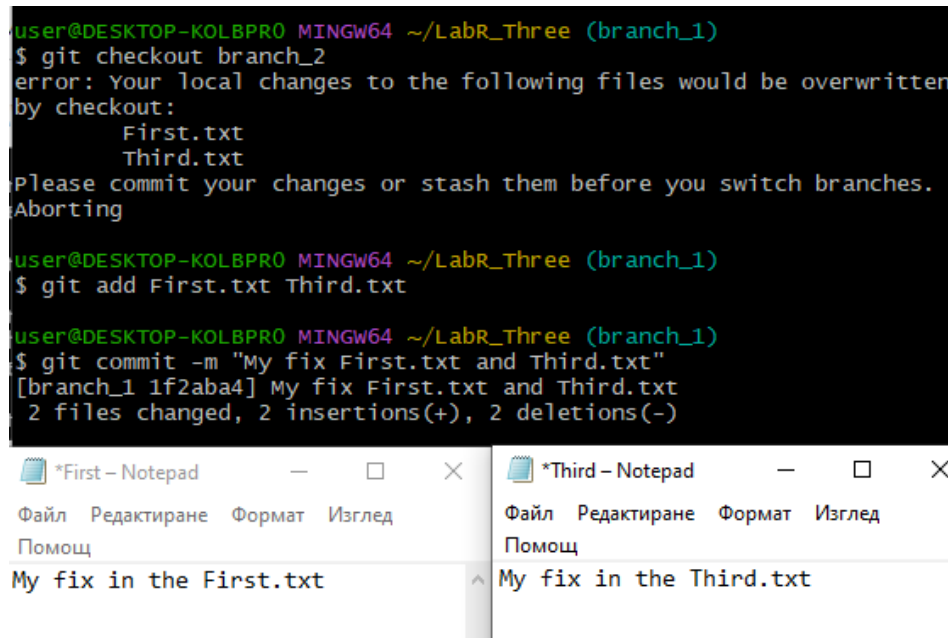
user@DESKTOP-KOLBPR0 MINGW64 ~/LabR_Three (branch_1)
$ git commit -m "changed First.txt and Third.txt"
[branch_1 1f4d009] changed First.txt and Third.txt
4 files changed, 3 insertions(+)
create mode 100644 In_Branch.txt
create mode 100644 Second.txt
create mode 100644 Third.txt
```



The screenshot shows two Notepad windows. The left window, titled 'First - Notepad', contains the text 'fix in the First.txt'. The right window, titled 'Third - Notepad', contains the text 'fix in the Third.txt'.

Рисунок 3.11 – Переход на ветку branch_1, изменение файлов

12. Перейти на ветку `branch_2` и также изменить файл `1.txt`, удалить все содержимое и добавить текст “My fix in the 1.txt”, изменить файл `3.txt`, удалить все содержимое и добавить текст “My fix in the 3.txt”, закоммитить изменения.



```
user@DESKTOP-KOLBPRO MINGW64 ~/LabR_Three (branch_1)
$ git checkout branch_2
error: Your local changes to the following files would be overwritten
by checkout:
      First.txt
      Third.txt
Please commit your changes or stash them before you switch branches.
Aborting

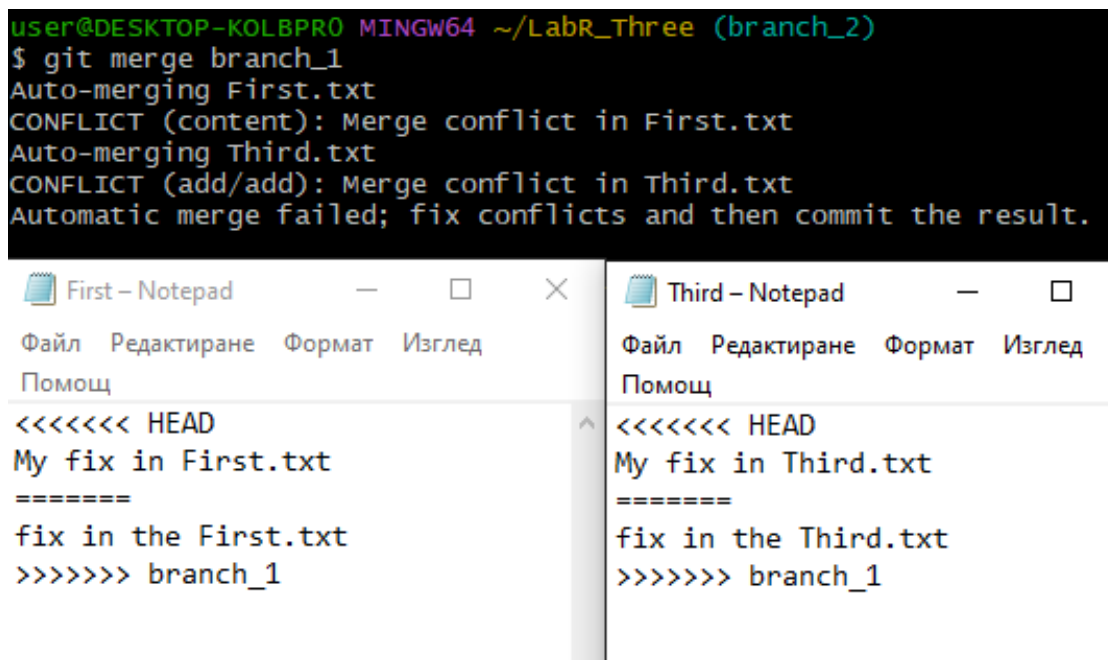
user@DESKTOP-KOLBPRO MINGW64 ~/LabR_Three (branch_1)
$ git add First.txt Third.txt

user@DESKTOP-KOLBPRO MINGW64 ~/LabR_Three (branch_1)
$ git commit -m "My fix First.txt and Third.txt"
[branch_1 1f2aba4] My fix First.txt and Third.txt
2 files changed, 2 insertions(+), 2 deletions(-)
```

The terminal window shows the user attempting to checkout `branch_2` but being blocked by local changes. After committing the changes, the terminal shows the commit message and file changes. Below the terminal, two Notepad windows are open: `*First - Notepad` and `*Third - Notepad`. Both windows show the text "My fix in the First.txt" and "My fix in the Third.txt" respectively.

Рисунок 3.12 – Переход на ветку `branch_2` и изменение файлов

13. Слить изменения ветки `branch_2` в ветку `branch_1`.



```
user@DESKTOP-KOLBPRO MINGW64 ~/LabR_Three (branch_2)
$ git merge branch_1
Auto-merging First.txt
CONFLICT (content): Merge conflict in First.txt
Auto-merging Third.txt
CONFLICT (add/add): Merge conflict in Third.txt
Automatic merge failed; fix conflicts and then commit the result.
```

The terminal window shows the user attempting to merge `branch_1` into `branch_2`. The merge fails due to conflicts in `First.txt` and `Third.txt`. Below the terminal, two Notepad windows are open: `First - Notepad` and `Third - Notepad`. Both windows show the text "My fix in First.txt" and "My fix in Third.txt" respectively, with a conflict marker "=====" between the two lines of text.

Рисунок 3.13 – Результат слияния веток

14. Решить конфликт файла 1.txt в ручном режиме, а конфликт 3.txt используя команду `git mergetool` с помощью одной из доступных утилит, например Meld.

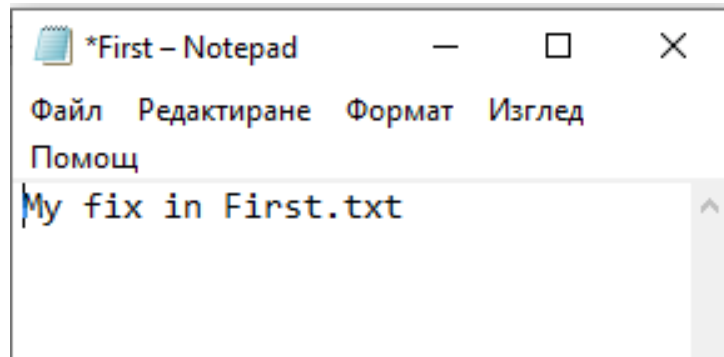


Рисунок 3.14 – Решение конфликта файлов вручную

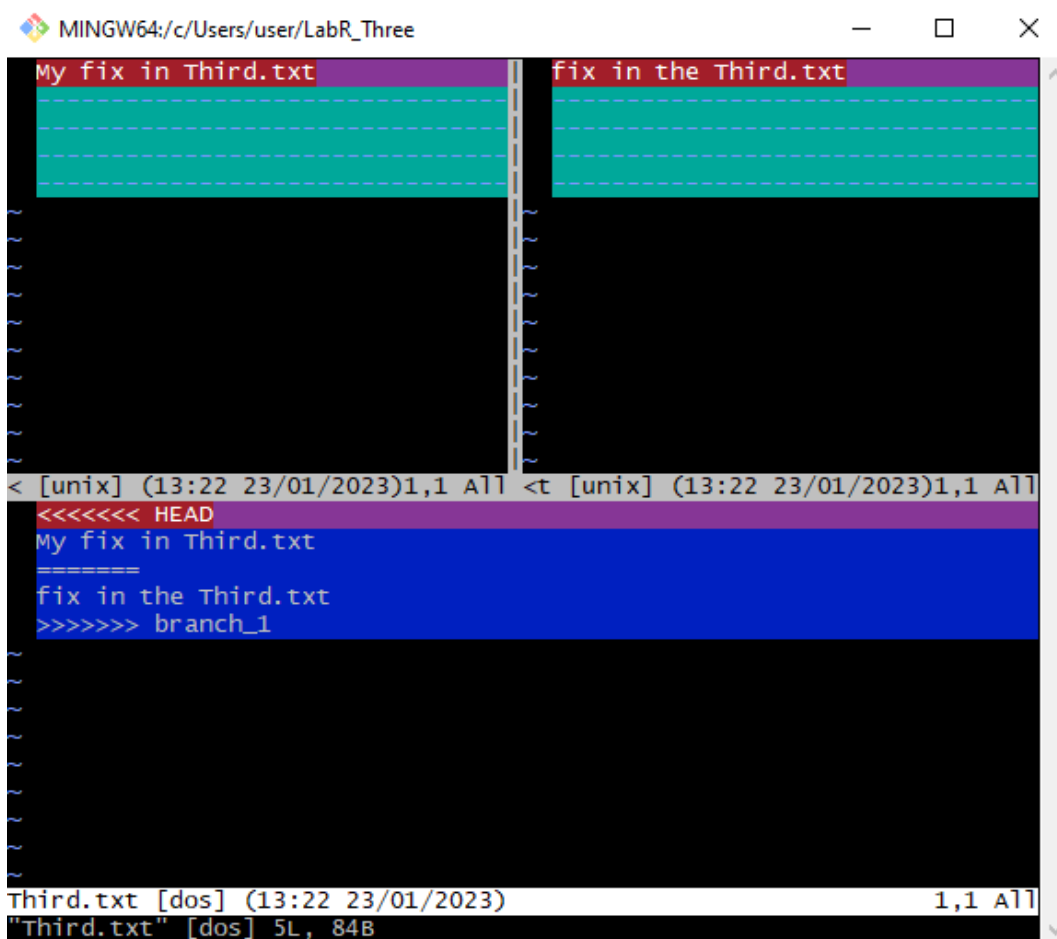


Рисунок 3.15 – Решение конфликта файлов через `git mergetool` с использованием `vimdiff`

```

$ git mergetool

This message is displayed because 'merge.tool' is not configured.
See 'git mergetool --tool-help' or 'git help config' for more details.
'git mergetool' will now attempt to use one of the following tools:
opendiff kdiff3 tkdiff xxdiff meld tortoisemerge gvimdiff diffuze diff
merge ecmerge p4merge araxis bc codecompare smerge emerge vimdiff nvim
diff
Merging:
First.txt
Third.txt

Normal merge conflict for 'First.txt':
 {local}: modified file
 {remote}: modified file
Hit return to start merge resolution tool (vimdiff): vimdiff
4 files to edit

$Normal merge conflict for 'Third.txt':
 {local}: created file
 {remote}: created file
user@DESKTOP-KOLBPR0 MINGW64 ~/LabR_Three (branch_2|MERGING)
$ |

```

Рисунок 3.16 – Результат работы git mergetool

15. Отправить ветку branch_1 на GitHub.

```

user@DESKTOP-KOLBPR0 MINGW64 ~/LabR_Three (branch_2|MERGING)
$ git push origin branch_1
Enumerating objects: 20, done.
Counting objects: 100% (20/20), done.
Delta compression using up to 2 threads
Compressing objects: 100% (11/11), done.
Writing objects: 100% (19/19), 1.55 kiB | 54.00 kiB/s, done.
Total 19 (delta 5), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (5/5), done.
remote:
remote: Create a pull request for 'branch_1' on Github by visiting:
remote:   https://github.com/DGalbacheva/LabR_Three/pull/new/branch_1
remote:
To https://github.com/DGalbacheva/LabR_Three.git
 * [new branch]      branch_1 -> branch_1

```

Рисунок 3.17 – Отправка ветки на GitHub

16. Создать средствами GitHub удаленную ветку branch_3.

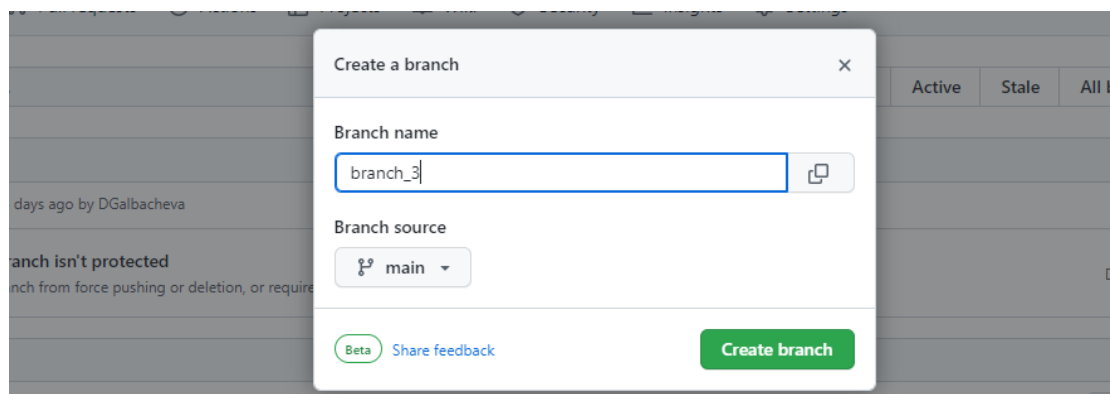


Рисунок 3.18 – Создание удаленной ветки через GitHub

17. Создать в локальном репозитории ветку отслеживания удаленной ветки branch_3.

```
MINGW64:/c/Users/user/LabR_Three
gi
user@DESKTOP-KOLBPR0 MINGW64 ~/LabR_Three (branch_2)
$ git checkout --track origin/branch_3
Switched to a new branch 'branch_3'
branch 'branch_3' set up to track 'origin/branch_3'.

user@DESKTOP-KOLBPR0 MINGW64 ~/LabR_Three (branch_3)
$
```

Рисунок 3.19 – Создание веток отслеживания

18. Перейти на ветку branch_3 и добавить файл файл 2.txt строку "the final fantasy in the 4.txt file".

```
user@DESKTOP-KOLBPR0 MINGW64 ~/LabR_Three (branch_2)
$ git checkout --track origin/branch_3
Switched to a new branch 'branch_3'
branch 'branch_3' set up to track 'origin/branch_3'.

user@DESKTOP-KOLBPR0 MINGW64 ~/LabR_Three (branch_3)
$
```

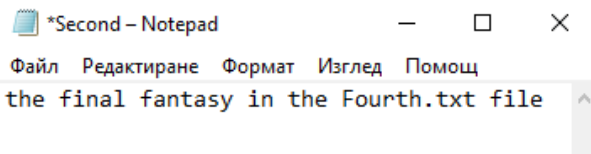


Рисунок 3.20 – Переход на ветку и добавление файла

19. Отправить изменения веток master и branch_2 на GitHub.

```
user@DESKTOP-KOLBPR0 MINGW64 ~/LabR_Three (branch_3)
$ git push origin main
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/DGalbacheva/LabR_Three.git
    b185a16..e06eb19  main -> main

user@DESKTOP-KOLBPR0 MINGW64 ~/LabR_Three (branch_3)
$ git push origin branch_2
Enumerating objects: 15, done.
Counting objects: 100% (15/15), done.
Delta compression using up to 2 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (10/10), 905 bytes | 33.00 KiB/s, done.
Total 10 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
remote:
remote: Create a pull request for 'branch_2' on Github by visiting:
remote:   https://github.com/DGalbacheva/LabR_Three/pull/new/branch_2
remote:
To https://github.com/DGalbacheva/LabR_Three.git
 * [new branch]      branch_2 -> branch_2
```

Рисунок 3.21 – Отправление изменения веток на GitHub

Ответы на вопросы:

1. Что такое ветка?

Ветка в Git – это просто легковесный подвижный указатель на один из коммитов.

2. Что такое HEAD?

HEAD - это указатель, задача которого ссылаться на определенный коммит в репозитории.

HEAD – это указатель на коммит в вашем репозитории, который станет родителем следующего коммита.

3. Способы создания веток.

Командой `git branch`

Командой `git checkout -b`

4. Как узнать текущую ветку?

Командой `git branch`

5. Как переключаться между ветками?

`Git checkout «имя ветки»`

6. Что такое удаленная ветка?

Удаленные ветки – это ссылки на состояние веток в удаленных репозиториях.

7. Что такое ветка отслеживания?

Ветки сложения – это ссылки на определенное состояние удаленных веток.

8. Как создать ветку отслеживания?

Для синхронизации ваших изменений с удаленным сервером выполним команду `git fetch origin`. Далее прописать `git checkout –track origin/«имя ветки»`

9. Как отправить изменения из локальной ветки в удаленную ветку?

Командой `git push «имя ветки»`

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

Git pull – это, по сути, команда git fetch, после которой сразу же следует get merge. Get fetch получает изменения в локальную копию.

11. Как удалить локальную и удаленную ветки?

Удаление удаленной ветки производится при помощи команды: git push origin –delete «имя ветки»

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?

Git-flow – альтернативная модель ветвления Git, в которой используются функциональные ветки и несколько основных веток.

Под каждую новую функцию нужно выделить собственную ветку, которую можно отправить в центральный репозиторий для создания резервной копии или совместной работы команды. Ветки feature создаются не на основе main, а на основе develop. Когда работа над функцией завершается, соответствующая ветка сливается с веткой develop. Функции не следует отправлять напрямую в ветку main.

Последовательность действий при работе по модели Gitflow:

1. Из ветки main создается ветка develop.
2. Из ветки develop создается ветка release.
3. Из ветки develop создаются ветки feature.
4. Когда работа над веткой feature завершается, она сливается в ветку develop.
5. Когда работа над веткой release завершается, она сливается с ветками develop и main.
6. Если в ветке main обнаруживается проблема, из main создается ветка hotfix.
7. Когда работа над веткой hotfix завершается, она сливается с ветками develop и main.

Первая проблема: авторам приходится использовать ветку develop вместо master, поскольку master зарезервирован для кода, который отправляется в продакшен. Существует сложившийся обычай называть рабочую ветвь по умолчанию master, и делать ответвления и слияния с ней. Большинство инструментов по умолчанию используют это название для

основной ветки и по умолчанию выводят именно ее, и бывает неудобно постоянно переключаться вручную на другую ветку.

Вторая проблема процесса git flow – сложности, возникающие из-за веток для патчей и для релиза. Подобная структура может подойти некоторым организациям, но для абсолютного большинства она просто убийственно излишняя. На сегодняшний день большинство компаний практикуют непрерывное разветвление (continuous delivery), что подразумевает, что основная ветвь по умолчанию может быть задеплоена (deploy). А значит, можно избежать использования веток для релиза и патчей, и всех связанных с нами хлопот, например, обратного слияния из веток релизов.

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.

