

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ
ФЕДЕРАЦИИ**
**Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

Кафедра инфокоммуникаций

**«Исследование возможностей Git для работы с
локальными репозиториями»**

Отчет по лабораторной работе № 1.3

по дисциплине «Основы программной инженерии»

Выполнил студент группы ПИЖ-б-о-21-1

Трушева В. О. « » сентября 2022г.

Подпись студента _____

Работа защищена « » _____ 20__ г.

Проверила Воронкин Р.А. _____
(подпись)

Ставрополь 2022

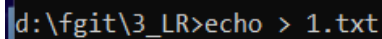
Цель работы: исследование базовых возможностей по работе с локальными и удаленными ветками Git.

Методика и порядок выполнения работы

1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT

Рисунок 1 – Создание репозитория

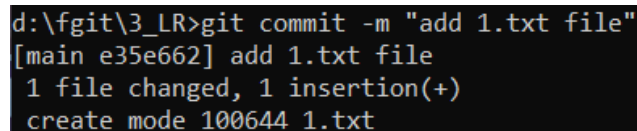
3. Создать три файла: 1.txt, 2.txt, 3.txt.



```
d:\fgit\3_LR>echo > 1.txt
```

Рисунок 2 – Создание текстового файла

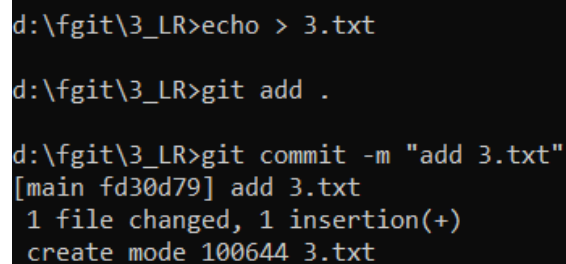
4. Проиндексировать первый файл и сделать коммит с комментарием "add 1.txt file".



```
d:\fgit\3_LR>git commit -m "add 1.txt file"
[main e35e662] add 1.txt file
1 file changed, 1 insertion(+)
create mode 100644 1.txt
```

Рисунок 3 – Коммит с комментарием о добавление нового файла

5. Проиндексировать второй и третий файлы.



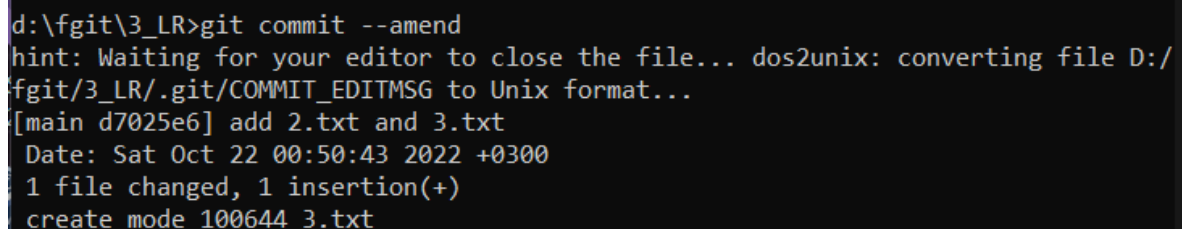
```
d:\fgit\3_LR>echo > 3.txt

d:\fgit\3_LR>git add .

d:\fgit\3_LR>git commit -m "add 3.txt"
[main fd30d79] add 3.txt
1 file changed, 1 insertion(+)
create mode 100644 3.txt
```

Рисунок 4 – Коммит последнего файла

6. Переписать уже сделанный коммит с новым комментарием "add 2.txt and 3.txt."



```
d:\fgit\3_LR>git commit --amend
hint: Waiting for your editor to close the file... dos2unix: converting file D:/fgit/3_LR/.git/COMMIT_EDITMSG to Unix format...
[main d7025e6] add 2.txt and 3.txt
Date: Sat Oct 22 00:50:43 2022 +0300
1 file changed, 1 insertion(+)
create mode 100644 3.txt
```

Рисунок 5 – Перезапись коммита

7. Создать новую ветку my_first_branch.

```
d:\fgit\3_LR>git branch my_first_branch

d:\fgit\3_LR>git log
commit d7025e698d047902ac22f14785c0322d6d0eb4e7 (HEAD -> main, my_first_branch)
Author: pynikcmd <ver.r762@gmail.com>
Date: Sat Oct 22 00:50:43 2022 +0300

    add 2.txt and 3.txt
```

Рисунок 6 – Создание новой ветки

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

```
d:\fgit\3_LR>git checkout my_first_branch
Switched to branch 'my_first_branch'
```

Рисунок 7 – Переход на ветку my_first_branch

```
d:\fgit\3_LR>echo > in_branch.txt

d:\fgit\3_LR>git add in_branch.txt

d:\fgit\3_LR>git commit -m "add in_branch.txt"
[my_first_branch e4c8750] add in_branch.txt
1 file changed, 1 insertion(+)
create mode 100644 in_branch.txt
```

Рисунок 8 – Создание нового файла с коммитом

9. Вернуться на ветку main.

```
d:\fgit\3_LR>git checkout main
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 2 commits.
(use "git push" to publish your local commits)
```

Рисунок 9 – Возвращение на ветку main

10. Создать и сразу перейти на ветку new_branch.

```
d:\fgit\3_LR>git checkout -b new_branch
Switched to a new branch 'new_branch'
```

Рисунок 10 – Создание и переход на новую ветку

11. Сделать изменения в файле 1.txt, добавить строчку “new row in the 1.txt file”, закоммитить изменения.

```
d:\fgit\3_LR>echo "new row in the 1.txt file" >> 1.txt
```

Рисунок 11 – Добавление строчки в файл

```
d:\fgit\3_LR>git commit -m "add row in the 1.txt"
[new_branch 661724d] add row in the 1.txt
1 file changed, 1 insertion(+)
```

Рисунок 12 – Первый коммит в ветке new_branch

12. Перейти на ветку main и слить ветки main и my_first_branch, после чего слить ветки main и new_branch.

```
d:\fgit\3_LR>git checkout main
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 2 commits.
(use "git push" to publish your local commits)

d:\fgit\3_LR>git merge my_first_branch
Updating d7025e6..e4c8750
Fast-forward
 in_branch.txt | 1 +
1 file changed, 1 insertion(+)
 create mode 100644 in_branch.txt

d:\fgit\3_LR>git branch
* main
  my_first_branch
  new_branch

d:\fgit\3_LR>git merge new_branch
Merge made by the 'ort' strategy.
1.txt | 1 +
1 file changed, 1 insertion(+)
```

Рисунок 13 – Результаты слияния веток

13. Удалить ветки my_first_branch и new_branch.

```
d:\fgit\3_LR>git branch -d my_first_branch
Deleted branch my_first_branch (was e4c8750).

d:\fgit\3_LR>git branch -d new_branch
Deleted branch new_branch (was 661724d).

d:\fgit\3_LR>git branch
* main
```

Рисунок 14 – Результат удаления веток

14. Создать ветки branch_1 и branch_2.

```
d:\fgit\3_LR>git branch branch_1

d:\fgit\3_LR>git branch branch_2

d:\fgit\3_LR>git branch
branch_1
branch_2
* main
```

Рисунок 15 – Результат создания веток branch_1 и branch_2

15. 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.

```
d:\fgit\3_LR>git checkout branch_1
Switched to branch 'branch_1'

d:\fgit\3_LR>echo . > 1.txt

d:\fgit\3_LR>echo "fix in the 1.txt" > 1.txt

d:\fgit\3_LR>echo "fix in the 3.txt" > 3.txt

d:\fgit\3_LR>git add .

d:\fgit\3_LR>git commit -m "changed text in 1.txt and 3.txt"
[branch_1 d0d2bef] changed text in 1.txt and 3.txt
2 files changed, 2 insertions(+), 3 deletions(-)
```

Рисунок 16 – Изменения в файлах и их коммит в ветке branch_1

16. 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.

```
d:\fgit\3_LR>git checkout branch_2
Switched to branch 'branch_2'
M      1.txt

d:\fgit\3_LR>echo "My fix in the 1.txt" > 1.txt

d:\fgit\3_LR>break 3.txt

d:\fgit\3_LR>echo "My fix in the 3.txt" > 3.txt

d:\fgit\3_LR>git add .

d:\fgit\3_LR>git commit -m "changed text in 1.txt and 3.txt"
[branch_2 341bbec] changed text in 1.txt and 3.txt
 2 files changed, 2 insertions(+), 3 deletions(-)
```

Рисунок 17 – Изменения в файлах и их коммит branch_2

17. Слить изменения ветки branch_2 в ветку branch_1.

```
d:\fgit\3_LR>git merge branch_1
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 18 – Результат слияния веток branch_2 и branch_1

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

```
d:\fgit\3_LR>git status
On branch branch_2
Your branch is up to date with 'origin/branch_2'.

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

d:\fgit\3_LR>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:
tortoisemerge emerge vimdiff nvimdiff
Merging:
3.txt
```

Рисунок 20 – Применение команды git mergetool

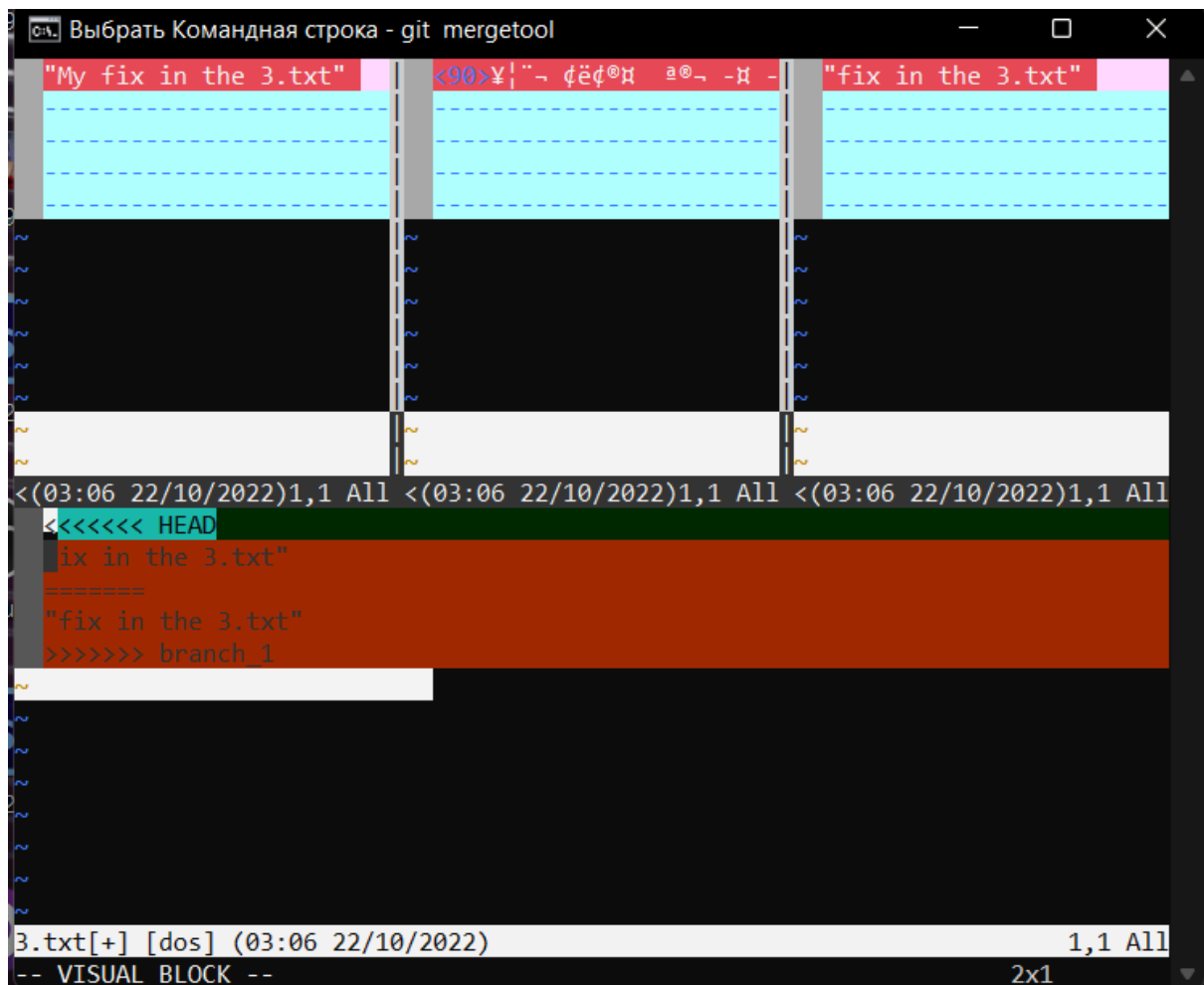


Рисунок 21 – Исправление ошибки

19. Отправить ветку branch_1 на GitHub.

```
d:\fgit\3_LR>git checkout branch_1
Switched to branch 'branch_1'

d:\fgit\3_LR>git push origin branch_1
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 20 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 397 bytes | 397.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'branch_1' on GitHub by visiting:
remote:   https://github.com/pynikcmd/3_LR/pull/new/branch_1
remote:
To https://github.com/pynikcmd/3_LR.git
 * [new branch]      branch_1 -> branch_1
```

Рисунок 22 – Отправка ветки на Git

20. Создать средствами GitHub удаленную ветку branch_3.

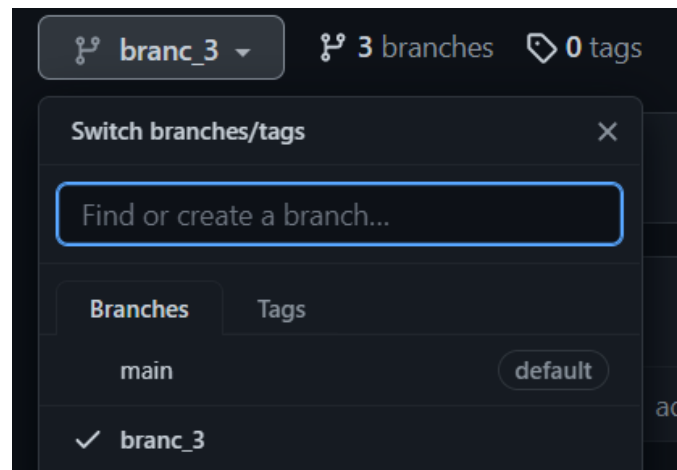


Рисунок 23 – Создание новой ветки в Git

```
d:\fgit\3_LR>git fetch --all
From https://github.com/pynikcmd/3_LR
* [new branch]      branc_3      -> origin/branc_3
```

Рисунок 24 – Получение данных со всех удаленных серверов

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

```
d:\fgit\3_LR>git checkout -b branc_3 origin/branc_3
Switched to a new branch 'branc_3'
branch 'branc_3' set up to track 'origin/branc_3'.
```

Рисунок 25 – Создание ветки слежения branc_3

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

```
d:\fgit\3_LR>echo "the final fantasy in 4.txt file" >> 2.txt
```

Рисунок 26 – Добавление текста в файл 2.txt

23. Выполнить перемещение ветки master на ветку branch_2.

```
d:\fgit\3_LR>git merge branch_2
Updating 38ec3ed..7a6a3c2
Fast-forward
 1.txt | 7 +++++--
 3.txt | 2 +-
 git   | 0
3 files changed, 6 insertions(+), 3 deletions(-)
create mode 100644 git
```

Рисунок 27 – Перемещение ветки master на ветку branch_2

24. Отправить изменения веток master и branch_2 на GitHub.

```
d:\fgit\3_LR>git push origin main
Everything up-to-date

d:\fgit\3_LR>git push origin branc_3
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 20 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 449 bytes | 449.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/pynikcmd/3_LR.git
 38ec3ed..7a6a3c2 branc_3 -> branc_3
```

Рисунок 28 – Последние коммиты

Вывод: в ходе лабораторной работы были исследованы базовые возможности системы контроля версий Git для работы с локальными репозиториями исследование базовых возможностей по работе с локальными и удаленными ветками Git.

Контрольные вопросы

1. Что такое ветка?

Ветка в Git — это простой перемещаемый указатель на один из таких коммитов. По умолчанию, имя основной ветки в Git — master.

2. Что такое HEAD?

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

3. Способы создания веток.

Создать ветку можно с помощью команды *git branch*.

4. Как узнать текущую ветку?

При помощи простой команды *git log*, которая покажет куда указывают указатели веток. Эта опция называется *--decorate*.

5. Как переключаться между ветками?

С помощью команды *git checkout*.

6. Что такое удаленная ветка?

Ветка, которая есть в удаленном репозитории, но нет на компьютере.

7. Что такое ветка отслеживания?

Ветки слежения — это локальные ветки, которые напрямую связаны с удалённой веткой.

8. Как создать ветку отслеживания?

Для синхронизации *git fetch origin*, а затем *git checkout --track origin/*.

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

Команда *git push origin*.

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

Команда *git fetch* получает с сервера все изменения, которых у вас ещё нет, но не будет изменять состояние вашей рабочей директории. Эта

команда просто получает данные и позволяет вам самостоятельно сделать слияние. Тем не менее, существует команда `git pull`, которая в большинстве случаев является командой `git fetch`, за которой непосредственно следует команда `git merge`

11. Как удалить локальную и удаленную ветки?

Удалить ветку на удаленном сервере можно, используя параметр `--delete` для команды `git push`. Для удаления ветки на сервере, выполнить следующую команду: `git push origin --delete`. Для локальной `git branch -d`.

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. В этом рабочем процессе для регистрации истории проекта вместо одной ветки `main` используются две ветки. В главной ветке `main` хранится официальная история релиза, а ветка разработки `develop` предназначена для объединения всех функций.

Когда в ветке `develop` оказывается достаточно функций для выпуска (или приближается назначенная дата релиза), от ветки `develop` создается ветка `release`. Создание этой ветки запускает следующий цикл релиза, и с этого момента новые функции добавить больше нельзя — допускается лишь исправление багов, 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.

Ветки сопровождения или исправления (hotfix) используются для быстрого внесения исправлений в рабочие релизы.