

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

Кафедра инфокоммуникаций

**Языки программирования
Отчет по лабораторной работе №1
«Основы ветвления Git»**

Выполнил студент группы

ИТС-б-о-20-1 (1)

Горлов Д. С. « » _____ 20__ г.

Подпись студента _____

Работа защищена « » _____ 20__ г.

Проверил к.т.н., доцент
Кафедры инфокоммуникаций
Воронкин Р.А.

(подпись)

Ставрополь 2021

Лабораторная работа №1

Основы ветвления Git

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

Ссылка на репозиторий:

https://github.com/gor-dimm/prog_lr1


Порядок выполнения работы:

1. Создан общедоступный репозиторий на GitHub, в котором использована лицензия MIT.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner *

 gor-dimm ▾

Repository name *

/ prog_lr1 ✓

Great repository names are short and memorable. Need inspiration? How about [animated-potato](#)?

Description (optional)

Языки программирования. Лабораторная работа №1

☒



Public

Anyone on the internet can see this repository. You choose who can commit.

☐



Private

You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

☒ Add a README file

This is where you can write a long description for your project. [Learn more.](#)


☐ Add .gitignore

Choose which files not to track from a list of templates. [Learn more.](#)

☒ Choose a license

A license tells others what they can and can't do with your code. [Learn more.](#)

License: MIT License ▾

This will set  main as the default branch. Change the default name in your [settings](#).

Create repository

Рисунок 1. Создание общедоступного репозитория

2. Добавлены файлы 1.txt, 2.txt и 3.txt.

GOR-DIMM (J:) > языки программирования > lp1 > prog_lr1 >

Имя	Дата изменения	Тип	Размер
.git	14.09.2021 13:44	Папка с файлами	
1.txt	14.09.2021 20:13	Текстовый докум...	1 КБ
2.txt	14.09.2021 20:13	Текстовый докум...	0 КБ
3.txt	14.09.2021 20:13	Текстовый докум...	0 КБ
LICENSE	14.09.2021 13:44	Файл	2 КБ
README.md	14.09.2021 13:44	Файл "MD"	1 КБ

Рисунок 2. Создание файлов

3. Проиндексирован первый файл и выполнен коммит.

```
(base) J:\языки программирования\lp1\prog_lr1>git add 1.txt
```

Рисунок 3. Индексация файла 1.txt

4. Проиндексированы второй и третий файлы, выполнен коммит.

```
(base) J:\языки программирования\lp1\prog_lr1>git add .
(base) J:\языки программирования\lp1\prog_lr1>git commit -m "added 2.txt and 3.txt"
```

Рисунок 4. Индексация файлов 2.txt и 3.txt

5. Создана новая ветка my_first_branch.

```
(base) J:\языки программирования\lp1\prog_lr1>git branch "my_first_branch"
```

Рисунок 5. Создание ветки

6. Выполнен переход на ветку my_first_branch, добавлен файл in_branch.txt, выполнен коммит.

7. Создана ветка new_branch и одновременно выполнен переход на неё.

```
(base) J:\языки программирования\lp1\prog_lr1>git checkout -b "new_branch"
```

Рисунок 6. Создание ветки и переход на неё в одной команде

8. Выполнены изменения в файле 1.txt, выполнен коммит.

9. Выполнено слияние веток master и my_first_branch, затем master и new_branch.

10. Удалены ветки my_first_branch и new_branch. Созданы ветки branch_1 и branch_2.

```
(base) J:\языки программирования\lp1\prog_lr1>git branch -d "my_first_branch"
(base) J:\языки программирования\lp1\prog_lr1>git branch -d "new_branch"
```

Рисунок 7. Удаление старых веток

11. Выполнен переход на branch_1, изменены файл 1.txt и 3.txt, выполнен КОММИТ.

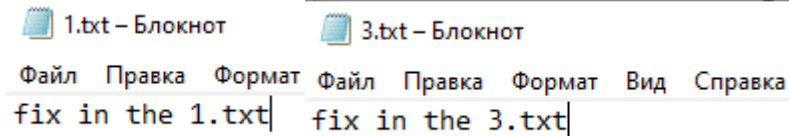


Рисунок 8. Изменения в файлах 1.txt и 3.txt

```
(base) J:\языки программирования\lp1\prog_lr1>git add 1.txt
(base) J:\языки программирования\lp1\prog_lr1>git add 3.txt
(base) J:\языки программирования\lp1\prog_lr1>git commit -m "changed 1.txt and 3.txt"
[branch_1 13af027] changed 1.txt and 3.txt
2 files changed, 2 insertions(+), 1 deletion(-)
```

Рисунок 9. Коммит изменений

12. Выполнен переход на ветку branch_2, изменены файл 1.txt и 3.txt, выполнен КОММИТ изменений.

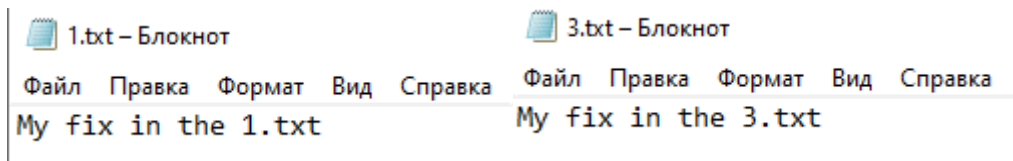


Рисунок 10. Изменение файлов

```
(base) J:\языки программирования\lp1\prog_lr1>git add 1.txt
(base) J:\языки программирования\lp1\prog_lr1>git add 3.txt
(base) J:\языки программирования\lp1\prog_lr1>git commit -m
error: switch `m' requires a value

(base) J:\языки программирования\lp1\prog_lr1>git commit
hint: Waiting for your editor to close the file... unix2dos: converting file J:\языки программирования\lp1\prog_lr1/.git
/COMMIT_EDITMSG to DOS format...
dos2unix: converting file J:\языки программирования\lp1\prog_lr1/.git/COMMIT_EDITMSG to Unix format...
Aborting commit due to empty commit message.

(base) J:\языки программирования\lp1\prog_lr1>git commit -m "changed 1.txt and 3.txt files"
[branch_2 f24e69e] changed 1.txt and 3.txt files
2 files changed, 2 insertions(+), 1 deletion(-)

(base) J:\языки программирования\lp1\prog_lr1>
```

Рисунок 11. Коммит файлов ветки 2

13. Выполнено слияние веток branch_1 и branch_2.

```
(base) J:\языки программирования\lr1\prog_lr1>git merge branch_2
Already up to date.
```

Рисунок 12. Слияние веток

14. Решён конфликт слияния файлов.

```
(base) J:\языки программирования\lr1\prog_lr1>git add .

(base) J:\языки программирования\lr1\prog_lr1>git checkout branch_1
Already on 'branch_1'
M       1.txt
M       3.txt

(base) J:\языки программирования\lr1\prog_lr1>git commit -m "fixed conflicts"
[branch_1 47f79d4] fixed conflicts
 2 files changed, 2 insertions(+), 2 deletions(-)
```

Рисунок 13. Коммит исправленного конфликта

15. Отправлена ветка branch_1 на GitHub.

```
(base) J:\языки программирования\lr1\prog_lr1>git push origin branch_1
info: please complete authentication in your browser...
Enumerating objects: 29, done.
Counting objects: 100% (29/29), done.
Delta compression using up to 4 threads
Compressing objects: 100% (18/18), done.
Writing objects: 100% (28/28), 2.40 KiB | 410.00 KiB/s, done.
Total 28 (delta 7), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (7/7), done.
remote:
remote: Create a pull request for 'branch_1' on GitHub by visiting:
remote:   https://github.com/gor-dimm/prog_lr1/pull/new/branch_1
remote:
To https://github.com/gor-dimm/prog_lr1
 * [new branch]   branch_1 -> branch_1
(base) J:\языки программирования\lr1\prog_lr1>
```

Рисунок 14. Отправление ветки

16. Создана удалённая ветка branch_3.

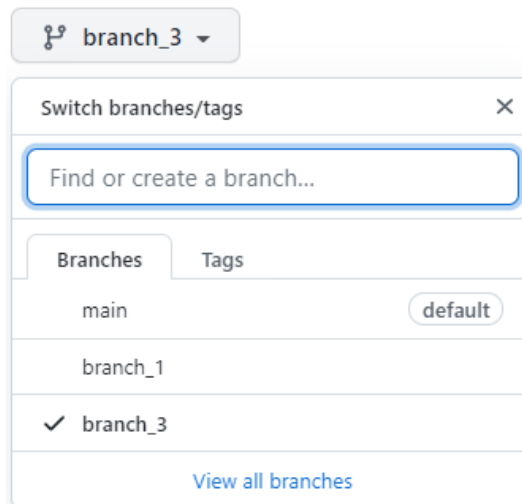


Рисунок 15. Создание удалённой ветки

17. Создана ветка отслеживания branch_3.

```
(base) J:\языки программирования\лр1\prog_lr1>git branch
* branch_1
  main

(base) J:\языки программирования\лр1\prog_lr1>git fetch origin
From https://github.com/gor-dimm/prog_lr1
* [new branch]      branch_3    -> origin/branch_3

(base) J:\языки программирования\лр1\prog_lr1>git checkout -b branch_3 origin/branch_3
Switched to a new branch 'branch_3'
Branch 'branch_3' set up to track remote branch 'branch_3' from 'origin'.

(base) J:\языки программирования\лр1\prog_lr1>
```

Рисунок 16. Создание ветки отслеживания

18. В ветке branch_3 изменён файл 2.txt.

19. Выполнено перемещение ветки master на branch_2, отправлены изменения на GitHub.

```
(base) J:\языки программирования\лр1\prog_lr1>git push origin
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 340 bytes | 340.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/gor-dimm/prog_lr1
    15826d6..683138e  branch_3 -> branch_3

(base) J:\языки программирования\лр1\prog_lr1>git push origin main
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/gor-dimm/prog_lr1
    15826d6..75462f9  main -> main

(base) J:\языки программирования\лр1\prog_lr1>git push
Everything up-to-date

(base) J:\языки программирования\лр1\prog_lr1>
```

Рисунок 17. Отправление всех изменений на GitHub

Контрольные вопросы:

1. Что такое ветка?

Ветка - простой перемещаемый указатель на один из нескольких родительских коммитов.

2. Что такое HEAD?

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

3. Способы создания веток.

Командами `git branch` или `git checkout`.

4. Как узнать текущую ветку?

Командой `git log` или командой `git branch` без указания каких-либо дополнительных параметров.

5. Как переключаться между ветками?

Командой `git checkout`.

6. Что такое удаленная ветка?

Удалённые ссылки — это ссылки (указатели) в удалённых репозиториях, включая ветки, теги и так далее.

7. Что такое ветка отслеживания?

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

8. Как создать ветку отслеживания?

Ветка отслеживания создаётся с помощью команды `git checkout -b *branch_name* origin/*branch_name*`

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

Изменения отправляются с помощью команды `git push *branch_name*`

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

Команда `git fetch` только получает данные с удалённого сервера, когда же команда `git pull` получает данные и выполняет слияние с веткой на рабочем месте.

11. Как удалить локальную и удалённую ветки?

Используя параметр `--delete` для команды `git push`.

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`?

Основные типы веток в модели `git-flow`:

- ветка разработки (`develop`);
- функциональная ветка (`feature`);
- ветка выпуска (`release`);
- ветка исправления (`hotfix`);

Работа с ветками организована следующим образом:

- из ветки `main` создается ветка `develop`;

- из ветки develop создается ветка release;
- из ветки develop создаются ветки feature;
- когда работа над веткой feature завершается, она сливается в ветку develop;
- когда работа над веткой release завершается, она сливается с ветками develop и main;
- если в ветке main обнаруживается проблема, из main создается ветка hotfix;
- когда работа над веткой hotfix завершается, она сливается с ветками develop и main.

Недостатками модели являются замедление работы, сложности с частотой релизов и трата времени в случае конфликтов.

Вывод о проделанной работе: проведено исследование базовых возможностей по работе с локальными и удаленными ветками Git.