

**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-22-1
Душин Александр Владимирович.
Подпись студента _____
Работа защищена « » _____ 20__ г.
Проверил Воронкин Р.А. _____
(подпись)

Ставрополь 2023

Тема: Основы ветвления Git.

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

Ход выполнения работы:


1. Создать общедоступный репозиторий на GitHub с использованием лицензии MIT:

Create a new repository

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

Required fields are marked with an asterisk (*).

Owner *

 MrPlatinum ▾

Repository name *

ProgrammEngineering3

✓ ProgrammEngineering3 is available.

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

Description (optional)



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:



Add a README file

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

Add .gitignore

.gitignore template: None ▾

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

Choose a license

License: MIT License ▾

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



You are creating a public repository in your personal account.

Create repository

Рисунок 1 – Создание общедоступного репозитория на GitHub с заданными настройками

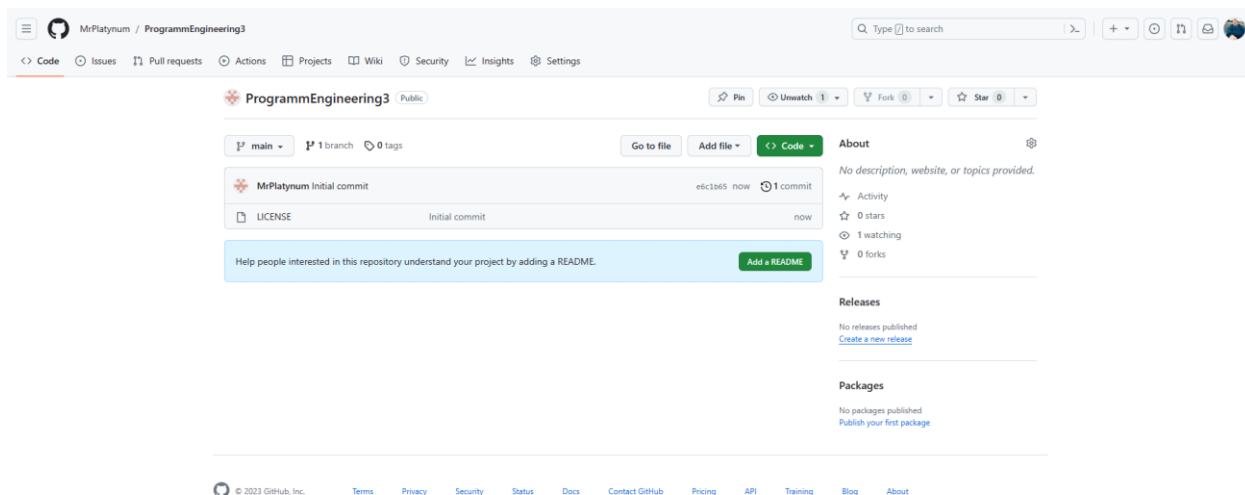


Рисунок 2 – Результат создания репозитория

2. Клонировать репозиторий на рабочий компьютер:

```
MINGW64:/c/Users/Alexander/Documents
Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents
$ git clone https://github.com/MrPlatynum/ProgrammEngineering3.git
Cloning into 'ProgrammEngineering3'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents
$
```

Рисунок 3 – Клонирование созданного репозитория на локальный компьютер

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

Имя	Дата изменения	Тип	Размер
1	04.11.2023 23:12	Текстовый докум...	0 КБ
2	04.11.2023 23:12	Текстовый докум...	0 КБ
3	04.11.2023 23:12	Текстовый докум...	0 КБ
LICENSE	04.11.2023 23:11	Файл	2 КБ

Рисунок 4 – Создание файлов

4. Проиндексировать первый файл и сделать коммит с комментарием «add 1.txt file»:

```
Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (main)
$ git add 1.txt

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (main)
$ git commit -m "add 1.txt"
[main 3bb6c85] add 1.txt
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 1.txt
```

Рисунок 5 – Индексация 1.txt и выполнение коммита

5. Проиндексировать второй и третий файлы:

```
Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (main)
$ git add 2.txt 3.txt
```

Рисунок 6 – Индексация оставшихся файлов

6. Перезаписать уже сделанный коммит с новым комментарием «add 2.txt and 3.txt»:

```
Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (main)
$ git commit --amend -m "add 2.txt and 3.txt"
[main 4991a30] add 2.txt and 3.txt
Date: Sat Nov 4 23:14:08 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 7 – Перезапись коммита с новым добавлением нового комментария

7. Создать новую ветку my_first_branch:

```
Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (main)
$ git branch my_first_branch
```

Рисунок 8 – Создание ветки my_first_branch

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

```
Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (main)
$ git checkout my_first_branch
Switched to branch 'my_first_branch'

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (my_first_br
nch)
$ touch in_branch.txt
```

Рисунок 9 – Переход на ветку my_first_branch и создание файла in_branch.txt

```
Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (my_
nch)
$ git add in_branch.txt

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (my_
nch)
$ git commit -m "add in_branch.txt"
[my_first_branch 96e0966] add in_branch.txt
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 in_branch.txt
```

Рисунок 10 – Коммит изменений

9. Вернуться на ветку master/main:

```
Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (my_fi
nch)
$ 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)
```

Рисунок 11 – Возвращение на ветку main

10. Создать и сразу перейти на ветку new_branch:

```
Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3
$ git checkout -b new_branch
Switched to a new branch 'new_branch'
```

Рисунок 12 – Создание ветки new_branch и переход в нее

11. Сделать изменения в файле 1.txt, добавить строчку «new row in the 1.txt file», закоммитить изменения:

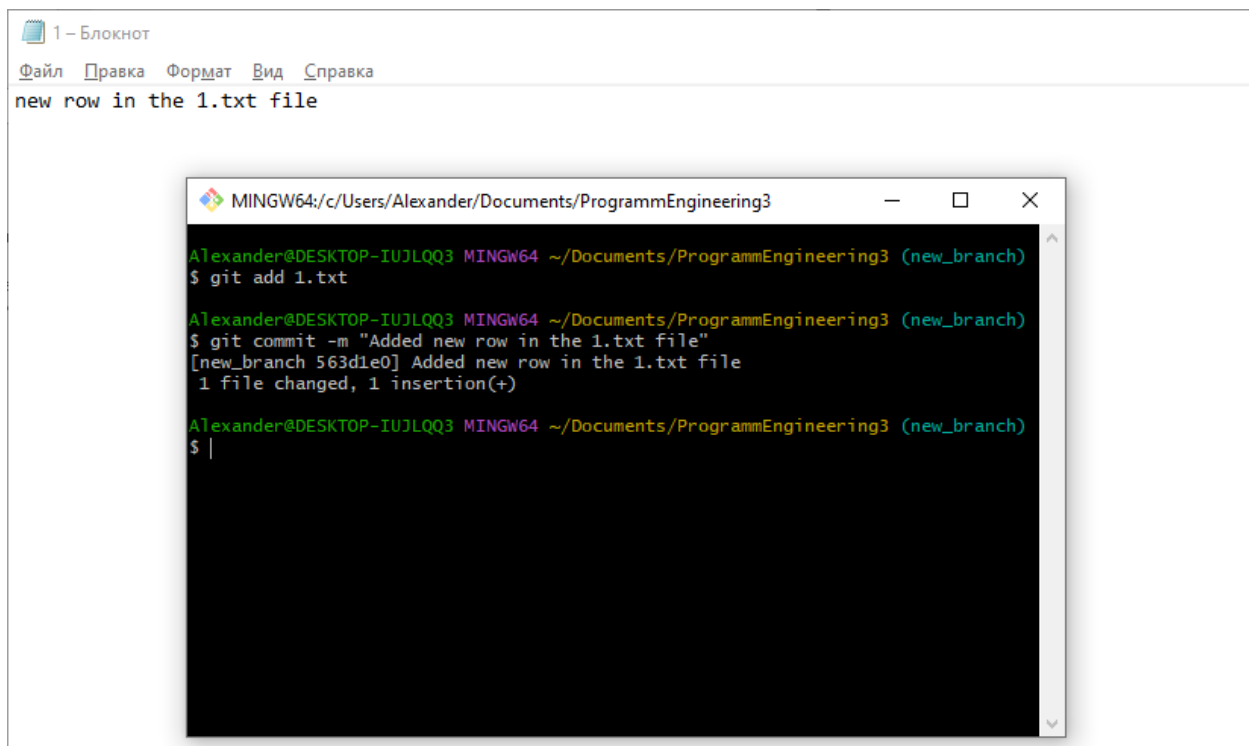


Рисунок 13 – Добавляем изменения в 1.txt и коммитим

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

```

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (main)
$ git checkout main
Already on 'main'
M       1.txt
Your branch is ahead of 'origin/main' by 2 commits.
  (use "git push" to publish your local commits)

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (main)
$ git merge my_first_branch
Already up to date.

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (main)
$ git checkout main
Already on 'main'
M       1.txt
Your branch is ahead of 'origin/main' by 2 commits.
  (use "git push" to publish your local commits)

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (main)
$ git merge new_branch
error: Your local changes to the following files would be overwritten by merge:
  1.txt
Merge with strategy ort failed.

```

Рисунок 14 – Переход на ветку main и слить с ней ветки my_first_branch и new_branch

13. Удалить ветки my_first_branch и new_branch:

```

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (main)
$ git branch -d my_first_branch
Deleted branch my_first_branch (was 0675e25).

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (main)
$ git branch -d new_branch
error: The branch 'new_branch' is not fully merged.
If you are sure you want to delete it, run 'git branch -D new_branch'.

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (main)
$ git branch -D new_branch
Deleted branch new_branch (was eb31700).

```

Рисунок 15 – Удаление веток my_first_branch и new_branch

14. Создать ветку branch_1 и branch_2:

```

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (main)
$ git branch branch_1

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (main)
$ git branch branch_2

```

Рисунок 16 – Создание 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:

```
Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (main)
$ git checkout branch_1
Switched to branch 'branch_1'
M       1.txt

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (branch_1)
$ echo "fix in the 1.txt" > 1.txt

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (branch_1)
$ echo "fix in the 3.txt" > 3.txt

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (branch_1)
$ git add 1.txt 3.txt
warning: in the working copy of '1.txt', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of '3.txt', LF will be replaced by CRLF the next time Git touches it

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (branch_1)
$ git commit -m "fixes in 1.txt and 3.txt"
[branch_1 f4d48ec] fixes in 1.txt and 3.txt
 2 files changed, 2 insertions(+)
```

Рисунок 17 – Переход на ветку `branch_1`, изменение в файлах `1.txt` и `3.txt` и коммит изменений

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:


```
Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (branch_1)
$ git checkout branch_2
Switched to branch 'branch_2'

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (branch_2)
$ echo "My fix in the 1.txt" > 1.txt

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (branch_2)
$ echo "My fix in the 3.txt" > 3.txt

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (branch_2)
$ git add 1.txt 3.txt
warning: in the working copy of '1.txt', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of '3.txt', LF will be replaced by CRLF the next time Git touches it

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (branch_2)
$ git commit -m "my fixes in 1.txt and 3.txt"
[branch_2 b174fef] my fixes in 1.txt and 3.txt
2 files changed, 2 insertions(+)
```

Рисунок 18 – Переход на ветку branch_2, изменение в файлах 1.txt и 3.txt и коммит изменений

17. Слить изменения ветки branch_2 в ветку branch_1:

```
Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (branch_2)
$ git checkout branch_1
Switched to branch 'branch_1'

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (branch_1)
$ git merge branch_2
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 19 – Конфликт соединения веток

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:

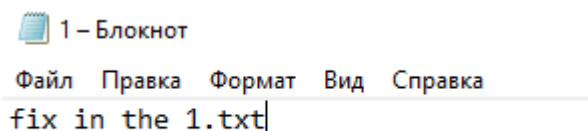


Рисунок 20 – Делаем вручную файл 1.txt таким же, как и в ветке branch_1

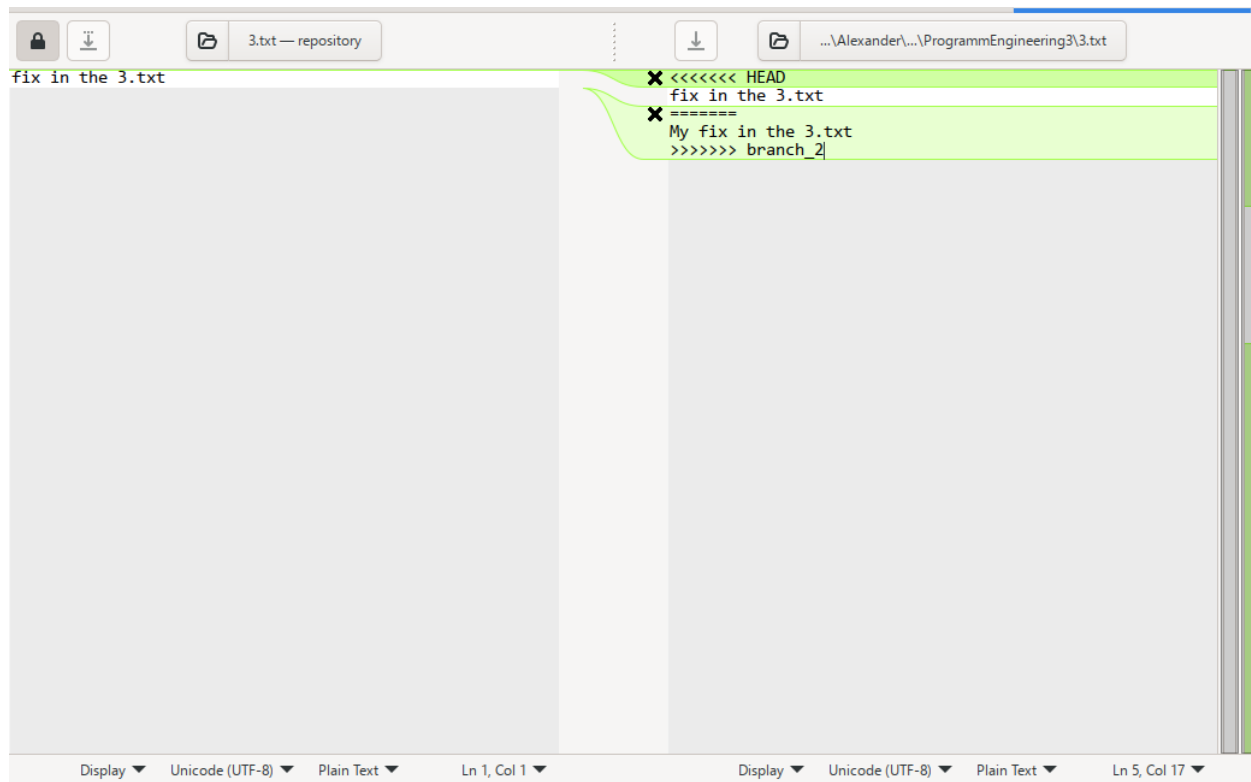


Рисунок 21 – Просмотр и решение конфликтов в файлах с помощью mergeTools

19. Отправить ветку branch_1 на GitHub:

```
Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (branch_1|MERGING)
$ git push origin branch_1
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 12 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (9/9), 787 bytes | 787.00 KiB/s, done.
Total 9 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), done.
remote:
remote: Create a pull request for 'branch_1' on GitHub by visiting:
remote:   https://github.com/MrPlatynum/ProgrammEngineering3/pull/new/branch_1
remote:
To https://github.com/MrPlatynum/ProgrammEngineering3.git
 * [new branch]      branch_1 -> branch_1

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (branch_1|MERGING)
$ |
```

Рисунок 22 – Отправка ветки branch_1 на удаленный репозиторий

20. Создать средствами GitHub удаленную ветку branch_3:

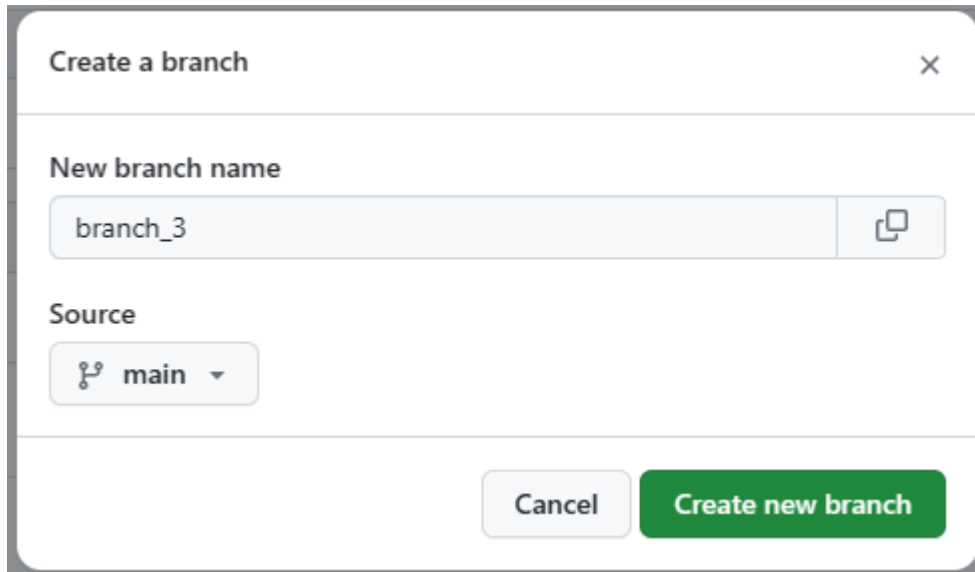


Рисунок 23 – Создание ветки branch_3, используя GitHub

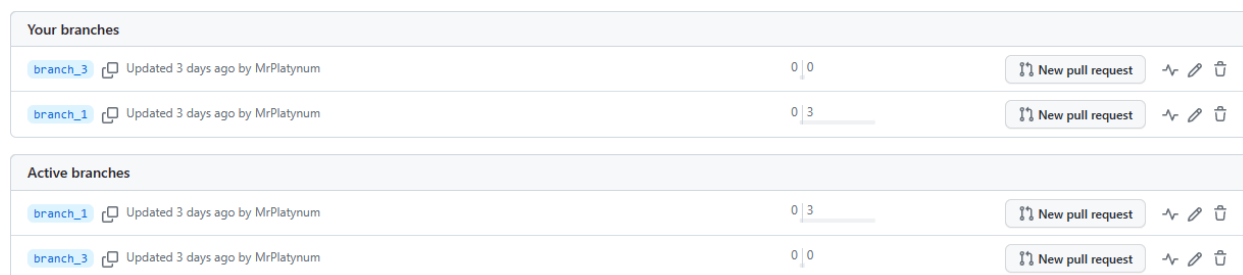


Рисунок 24 – Результат создания новой ветки

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

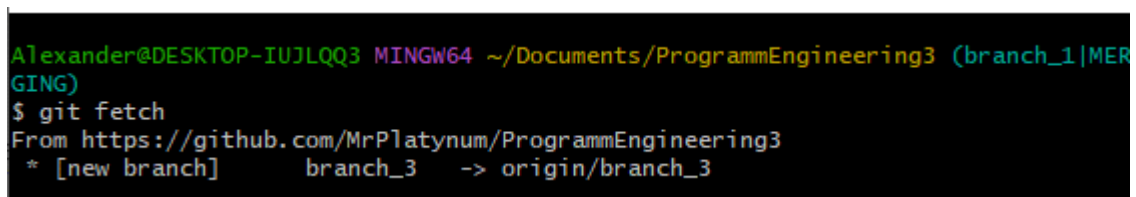


Рисунок 25 – Используя команду git fetch, обновится проект и автоматический создаться ветка branch_3, отслеживающая удаленную ветку origin/branch_3

22. Перейти на ветку branch_3 и добавить файл 2.txt в файл 2.txt строку «the final fantasy in the 4.txt file»:

```
Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (branch_1)
$ git checkout branch_3
Switched to a new branch 'branch_3'
branch 'branch_3' set up to track 'origin/branch_3'.
```

Рисунок 26 – Переход на ветку branch_3

```
Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (branch_3)
$ echo "the final fantasy in the 4.txt file" >> 2.txt
```

Рисунок 27 – Добавление строки в 2.txt

23. Выполнить перемещение ветки master/main на ветку branch_2:

```
Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (branch_3)
$ 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)

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (main)
$ git rebase branch_2
Successfully rebased and updated refs/heads/main.
```

Рисунок 28 – Переход на ветку main и ее перебазирование с веткой branch_2

24. Отправить изменения веток master/main и branch_2 на GitHub:

```
Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (main)
$ git push origin main
Enumerating objects: 7, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 403 bytes | 403.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/MrPlatynum/ProgrammEngineering3.git
   e6c1b65..b174fef  main -> main

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering3 (main)
$ git push origin branch_2
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'branch_2' on GitHub by visiting:
remote:   https://github.com/MrPlatynum/ProgrammEngineering3/pull/new/branch_2
remote:
To https://github.com/MrPlatynum/ProgrammEngineering3.git
 * [new branch]      branch_2 -> branch_2
```

Рисунок 29 – Отправка веток main/master и branch_2 на удаленный репозиторий

Ответы на контрольные вопросы:

1. Что такое ветка?

Ветка Git является перемещаемым указателем на коммиты.

2. Что такое HEAD?

HEAD – это указатель на коммит в вашей репозитории, который станет родителем следующего коммита. HEAD указывает на коммит, относительно которого будет создана рабочая копия во время операции checkout.

3. Способы создания веток.

Ветки можно создать следующими командами: `git branch <branch_name>` или `git checkout -b <branch_name>`.

4. Как узнать текущую ветку?

Воспользоваться командой `git branch` для вывода всех доступных веток. Если напротив названия ветки будет знак `*`, значит данная ветка является текущей.

5. Как переключаться между ветками?

Воспользоваться командой `git checkout <branch_name>`.

6. Что такое удаленная ветка?

Удалённые ветки – это ссылки на состояние веток в удалённом репозитории.

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. Представляйте их как закладки для напоминания о том, где ветки в удалённых репозиториях находились во время последнего подключения к ним.

8. Как создать ветку отслеживания?

Воспользовавшись командой `git fetch`, Git автоматически создаст ветки отслеживания с именами удалённых веток. Для создания своей ветки отслеживания можно воспользоваться командой `git checkout -b <branch>`

<remote>/<branch> или `git checkout --track <origin>/<branch>`. Если уже есть локальная ветка и нужно настроить ее на слежение за удаленной веткой, которую только что была получена, то: `git branch -u <origin>/<branch>`.

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

Для отправки изменения из локальной ветки в удаленную нужно воспользоваться командой `git push <remote> <branch>`.

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`. Если у вас настроена ветка слежения как показано в предыдущем разделе, или она явно установлена, или она была создана автоматически командами `clone` или `checkout`, `git pull` определит сервер и ветку, за которыми следит ваша текущая ветка, получит данные с этого сервера и затем попытается слить удалённую ветку.

11. Как удалить локальную и удаленную ветки?

Для удаления локальной ветки: `git branch -d <branch>`. Для удаленного репозитория: `git push <remote> --delete <branch>`.

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`?

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. Кроме того, для удобства рекомендуется присваивать всем коммитам в ветке `main` номер версии.

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.

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.

Когда подготовка к поставке завершается, релиз сливается с ветками main и develop, а ветка release удаляется. Важно слить ее с веткой develop, поскольку в ветку release могли добавить критические обновления, которые должны быть доступны для новых функций.

Ветки сопровождения или исправления (hotfix) используются для быстрого внесения исправлений в рабочие релизы. Ветки hotfix очень похожи на ветки release и feature. Отличие заключается в том, что 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.

Ключевые идеи, которые нужно запомнить о Gitflow:

- Данная модель отлично подходит для организации рабочего процесса на основе релизов.
- Работа по модели Gitflow предусматривает создание специальной ветки для исправления ошибок в рабочем релизе.

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`.

Рисунок 30 – Краткий итог Git-flow

Git-flow — это устаревшая версия рабочего процесса Git, в свое время ставшая принципиально новой стратегией управления ветками в Git. Популярность Git-flow стала снижаться под влиянием магистральных рабочих процессов, которые на сегодня считаются предпочтительными для современных схем непрерывной разработки ПО и применения DevOps. Кроме того, Git-flow не слишком удобно применять в процессах CI/CD.

При магистральной разработке программисты делают коммиты изменений прямо в основную ветку (мастер), а не создают отдельные функциональные ветки или ветки с исправлениями ошибок, которые объединяются с мастером позже.

При коммите изменений в основную ветку запускается CI/CD-пайплайн.

Если в пайплайне обнаруживаются ошибки, все сотрудники подключаются к работе, чтобы как можно скорее их устранить. Задача – поддерживать мастер в состоянии готовности к развертыванию, часто выпуская обновления.

CI/CD – одна из практик DevOps, подразумевающая непрерывную интеграцию и доставку. Этот набор принципов предназначен для повышения удобства, частоты и надежности развертывания изменений программного обеспечения или продукта.

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.

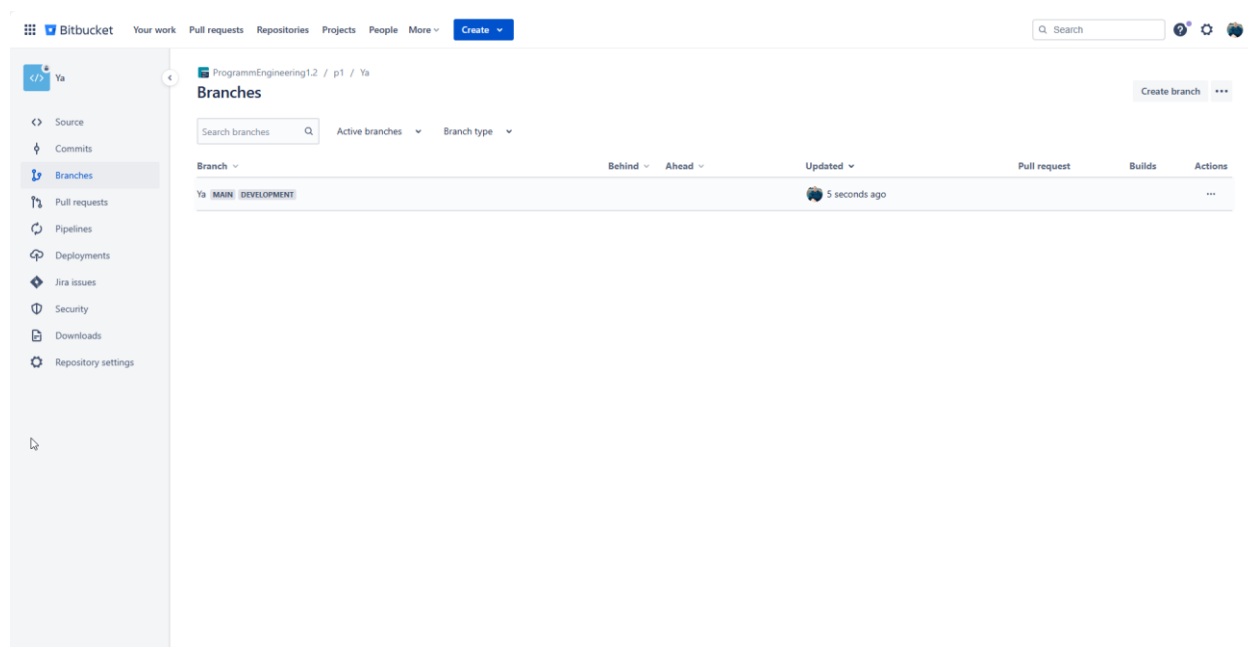


Рисунок 31 – Интерфейс показывает информацию о ветках, а также имеется возможность создания прямо из сервиса

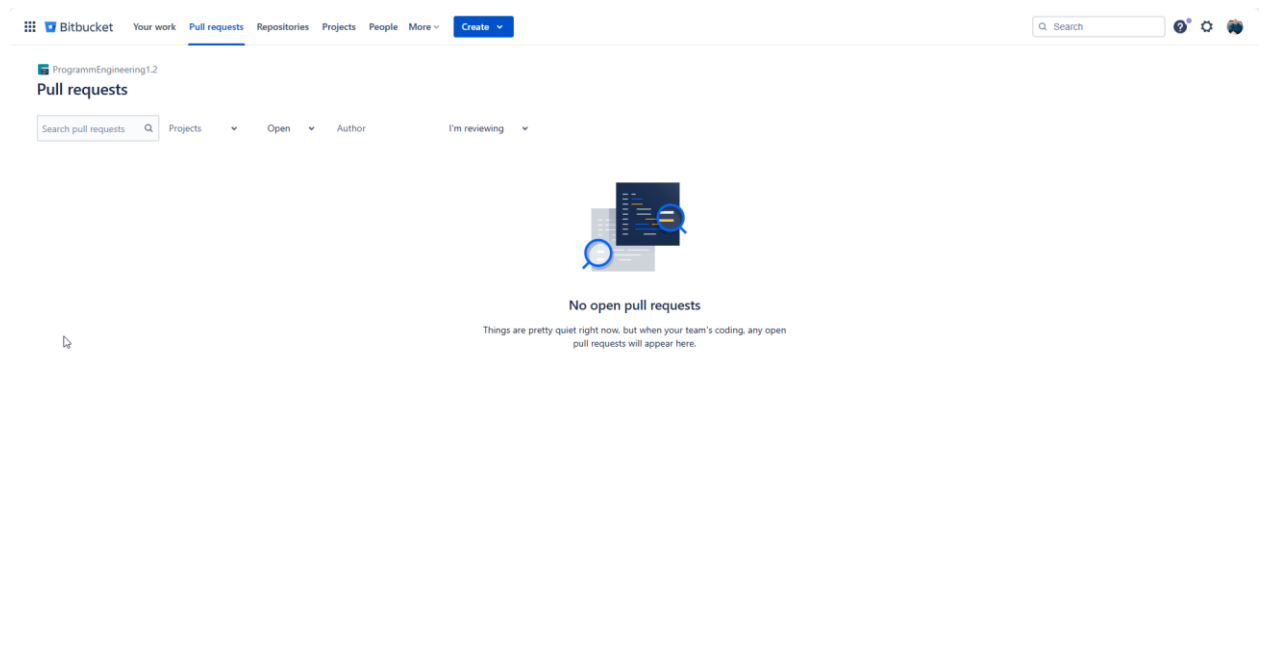


Рисунок 32 – Также можно отслеживать pull requests и т. д.