Министерство науки и высшего образования Российской Федерации

федеральное государственное бюджетное образовательное учреждение высшего образования

«Российский экономический университет имени Г.В. Плеханова»

**МОСКОВСКИЙ ПРИБОРОСТРОИТЕЛЬНЫЙ ТЕХНИКУМ**

специальность 09.02.07 «Информационные системы и программирование»

Квалификация: Программист

ПРАКТИЧЕСКИЕ РАБОТЫ

ПО МДК 04.02 «Обеспечение качества функционирования компьютерных систем»

|  |  |
| --- | --- |
| Выполнил студент  Группы П50-4-21  М. А. Замятин | Проверил преподаватель  \_\_\_\_\_\_\_\_\_\_\_ К.С. Образцова  «\_\_\_» \_\_\_\_\_\_\_\_\_ 2024 года |

Москва 2024

СОДЕРЖАНИЕ

[ЛЕКЦИОННЫЙ МАТЕРИАЛ 3](#_Toc183522682)

[ПРАКТИЧЕСКАЯ РАБОТА №1 10](#_Toc183522683)

# ЛЕКЦИОННЫЙ МАТЕРИАЛ

*03/09/2024*

**Waterfall** - каждый этап один за другим - нельзя вернуться назад. Этапы циркулируют только после завершения последнего этапа.



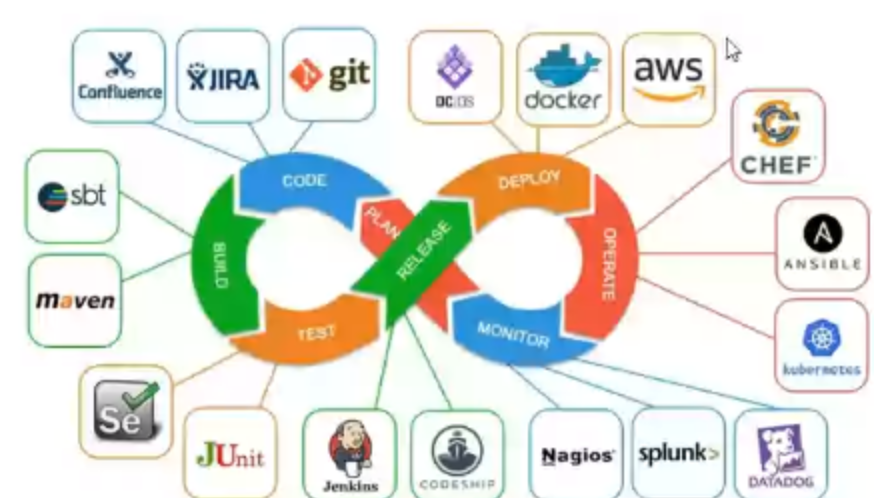
**SCRUM** - в конце спринта имеется готовый продукт. Гибкое изменение требований. Быстрое реагирование на изменение трендов. Бюджет не фиксирован.



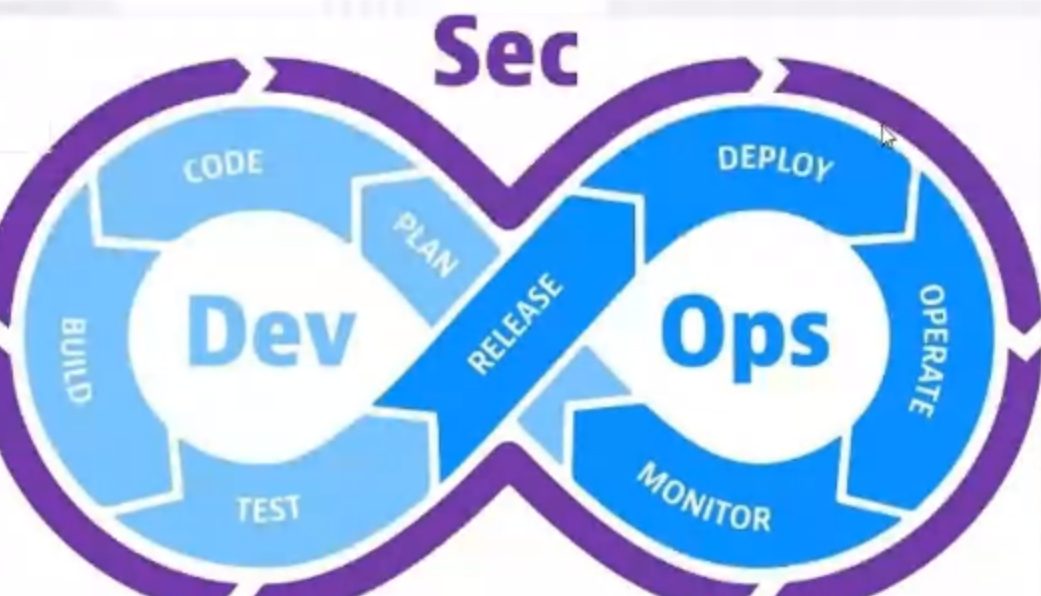
**Основные этапы разработки ПО:**

* Анализ;
* Разработка;
* Сборка;
* Тестирование;
* Развертывание;
* Релиз (деплой);
* Сопровождение;
* Мониторинг.

**DevOps (Development Operations)**- бесперебойная постановка ПО с помощью непрерывной интеграции рабочих процессов. Ускорение этапов разработки. Снижение ошибок, связанных с человеческим фактором.



**DevSecOps (Development Security Operations)** - интегрируется безопасность в процессы разработтки и автоматизации.



**Концепции состоявляющие DevOps:**

**Инфраструктура как код:**

Методология, предполагающая испольщование архитектурного подхода к управлению инфраструктурными ресурсами - автоматизация создания и управления инфраструктурными компонентами с помощью программного кода.

Цель: вся инфраструктура описывается в виде кода, который может быть управляемым исходными файлами.

**CI/CD:**

Методология разработки программного обеспечения, целью которой является автоматизация и ускорение процесса разработки, тестирования и доставки программного продукта.

Цель: быстрое время от идеи до развертывания новых функций или исправления ошибок, поддержка выского уровня качества кода и процесса разработки.

Концепция:

* Непрерывная интеграция (Countinuous Integration);
* Непрерывная доставка (Continuous Delivery);
* Непрерывное развертывание (Continuous Deployment).

**Преимущества внедрения DevOps в разработку:**

**Ускорение процесса разработки** - автоматизация и упрощение различных этапов разработки, сокращение времени, увеличение производительности.

**Улучшение качества ПО** - быстрое выявление и устранение ошибки и повышение стабильности и надежности продукта.

**Снижение рисков** - постоянный мониторинг и контроль процесса разработки.

*10/09/2024*

**Система контроля версий Git. GitLab и GitHub.**

**Система контроля версий** — это специальное программное обеспечение, которое используется для управления изменениями в файловой система, отслеживания и контроля версий документов или кода программы. Она позволяет разработчикам и команде проекта работать надд файлами вместе, отслеживать изменения, управлять их версиями и восстанавливаться к предыдущим состояниям проекта.

**Преимущества применения системы контроля версий:**

* История изменений;
* Контроль версий;
* Коллективная работа;
* Ветвление и слияние;
* Отслеживание ошибок.

**Распределенная VS Централизованная системы контроля версий.**

**Архитектура:**

* ЦСКВ имеет единую центральную БД, где храняться файлы и изменения.
* РСКВ пользователь имеет собственную копию репозитория.

**Работа в офлайн-режиме:**

* РСКВ позволяет работать в офлайн-режиме.
* ЦСКВ требуется подключение к центральному серверу для большинства операций.

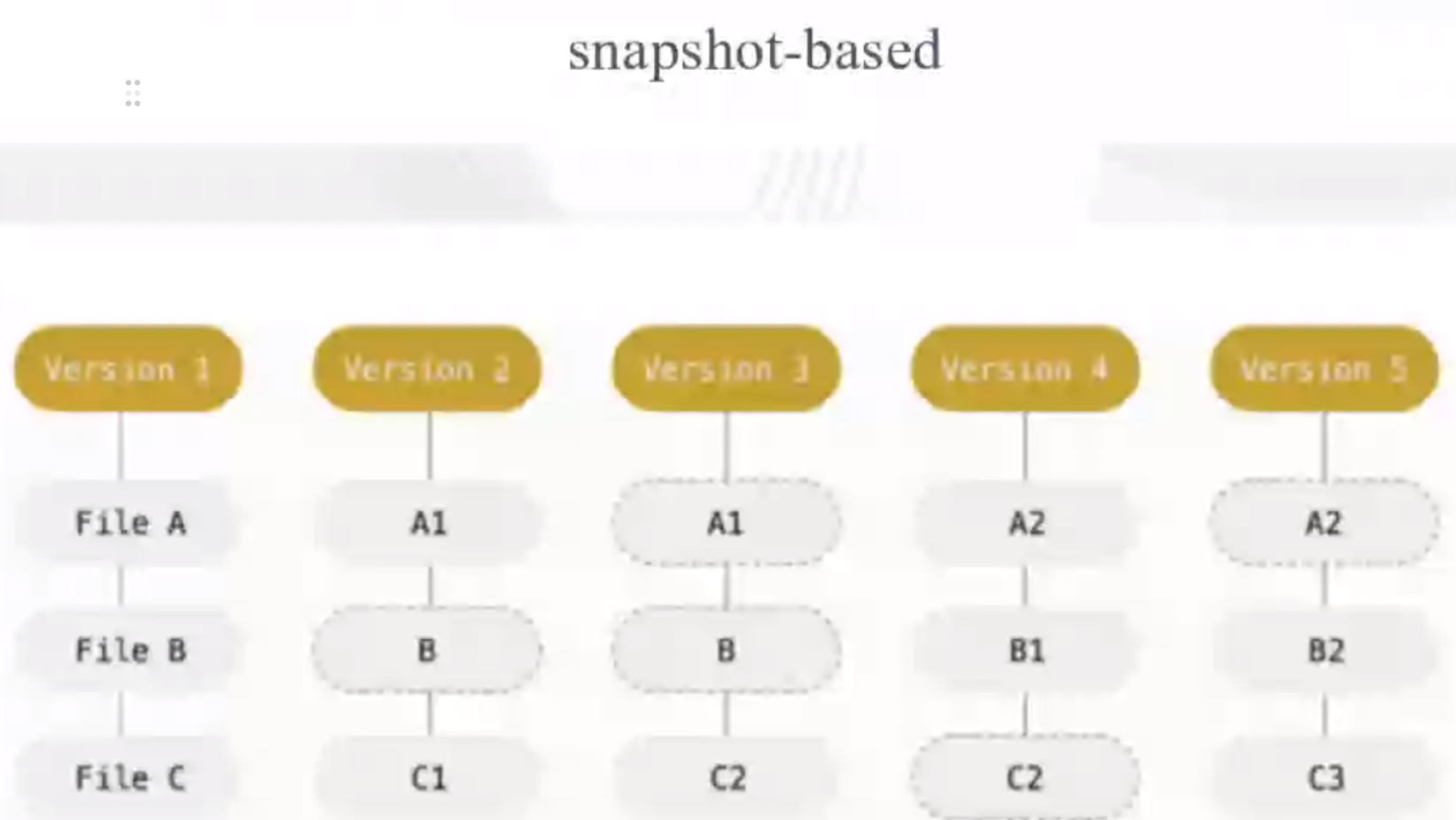
**Управление правами доступа:**

* ЦСКВ управление правами доступа происходит на уровне центрального сервера.
* РСКВ каждый пользователь может выполнять операции с собственной копией (более гибкое управление правами).

**Скорость работы:**

* РСКВ быстрее выполняет операции.
* ЦСКВ медленнее из-за удаленного соединения и большого количества пользователей.

**Система контроля версий с механизмом снимков** (Git, Mercurial) хранят все версии файлов и директорий в форме полных копий состояния проекта. При коммите система фиксирует измененные файлы и создает новые снимки - это мозволяет быстро переключаться между версиями проекта.



**Система контроля версий со списком изменений** (Subversion) хранят только разницу между последующими версиями файлов. После коммита система сохраняет только изменения, сделанные в файлах, относительно предыдущего коммита, в виде списка изменений. При переключении на предыдущие версии проекта, СКВ применяет все изменения последовательно, чтобы восстановить запрошенную версию проекта.



Системы контроля версий: Git, Perforce, Mercurial, Subversion.

**Коммит**

**Коммит (commit)** представляет собой операцию, при которой изменения в файловой системе, внесенные программистом, сохраняются в репозитории проекта. Коммит фиксирует изменения в файле или наборе файлов, а также добавляет комментарии или описания к этим изменениям. Каждый коммит имеет уникальный идентификатор, который позволяет отслеживать историю изменений и восстанавливать предыдущие версии файлов. Коммиты также позволяют работать с несколькими программистами над одним проектом, обьединяя изменения, внесенные каждым из них.

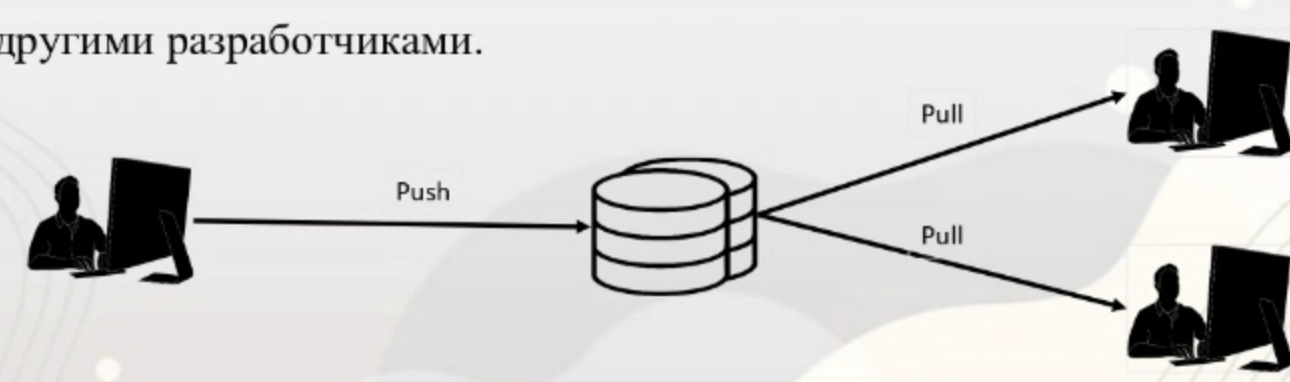
**Система контроля версий Git**

**Git** - РСКВ, которая используется для отслеживания изменений в коде ПО. Он позволяет разработчикам работать над проектом одновременно и совместно, сохраняя историю изменений и легко управляя версиями.



*27/09/2024*

**Git-хостинг** - это платформа, которые предоставляют инфраструктуру для хранения и управления репозиториями Git. Они позволяют разработчикам работать с Git репозиториями удаленно, управлять версиями, отслеживать изменения и сотрудничать с другими разработчиками.



**GitLab** - это хостинг-платформа для хранения и управления кодом. Она предоставляет функциональность, аналогичную GitHub, но с открытым исходным кодом. GitLab может быть развернут на собственных серверах, что позволяет организациям иметь полный контроль над своими репозиториями. Он также предлагает широкий набор инструментов для управления проектами.

**Введение в Docker**

**Docker** - это открытая платформа для разработки, доставки и запуска приложений. Он позволяет упаковать приложение и его зависимости в контейнер, который может быть запущен на любой совместимой с Docker системе. Контейнеры Docker предлагают независимую от операционной системы среду выполнения.

Преимущества использования Docker

**Изолированная среда**

Docker обеспечивает изолированное окружение, в котором приложение может работать независимо от других приложений и зависимостей на хостовой системе. Это позволяет избежать конфликтов и снижает вероятность неправильной работы приложения. Поэтому разработтчики могут не задумываться, в каком окружении будет работать их приложение, а инженеры по эксплуатации - единообразно запускать приложение и меньше заботиться о системных зависимостях.

**Управление зависимостями**

Docker позволяет включать все зависимости приложения, включая операционную систему, библиотеки и другие компоненты, в контейнере. Это упрощает управление зависимостями и обеспечивает согласованность окружения между разработчиками и на разных этапах разработки и может снизить проблемы, связанные с развертыванием приложения на разных серверах.

**Переносимость**

Docker позволяет легко перемещать контейнеры с приложением между различными средами разработки, тестирования и продакшн. Это упрощает процесс развертывания и масштабирования приложений.

**Dockerfile**

**Dockerfile** - это текстовый файл, содержащих инструкции для автоматического создания Docker-образа. Он описывает шаги, которые необходимо выполнить для создания контейнера Docker, включая установку и настройку приложений, копирование файлов и настройку сетей. Dockerfile позволяет разработчикам создавать повторяемые и легко воспроизводимые среды разработки и выполнения приложений, где каждый шаг указан явно и может быть автоматически выполнен при создании контейнера Docker.

**Containers(Контейнеры)**

**Контейнеры** - это изолированные и легковесные среды, в которых запускаются приложения и их зависимости. Они представляют собой запускаемые экземпляры образов Docker, содержащих все необходимое для работы приложения. Их можно создавать, запускать, останавливать и удалять. Также можно подключать к контейнеру хранилище, объединять контейнеры одной или несколькими сетями и общаться контейнеры одной или несколькими сетями и общаться с контейнерами, используя Docker API или командную строку.

1 контейнер = 1 приложение

Image (Образ)

Образ - шаблок контейнеров. В нем описывается, что должно быть установлено в контейнере (код, среда, библиотеки, окружение и тд) и какие действия нужно выполнить при старте контейнера.

Используя 1 образ можно создать много контейнеров.

**Docker Compose**

**Docker Compose** - инструмент, который позволяет запускать и управлять несколькими контейнерами одновременно. Позволяет описать структуру и конфигурацию вашего приложения в файле YAML, что делает его очень простым для понимания и использования. Может запустить и остановить все контейнеры одной командой, связать их между собой для взаимодействия.

**Docker демон**

**Docker демон** - основной компонент в системе Docker, который управляет выполнением и контролем контейнеров. Он работает в фоновом режиме и управляет ресурсами, созданием и удалением контейнеров, а также назначением им ресурсов.

**Docker клиент** - инструмент командной строки, который позволяет управлять контейнерами (через него взаимодействуют с демоном). Позволяет выполнять различные операции с контейнерами, такие как создание, запуск, остановка, удаление и мониторинг, используя простые команды в командной строке.

**Декларативное программирование** – это парадигма, при которой описывается желаемый результат, без составления детального алгоритма его получения.

**YAML (YAML Ain't Markup Language)** – это удобный формат для хранения и передачи данных в текстовой форме. Он используется для конфигурационных файлов, данных, и другой информации, которую читает и записывает компьютерное программное обеспечение. YAML напоминает человекочитаемый текст и позволяет организовать данные в виде списков, словарей и других структур для удобства чтения и редактирования как людьми, так и компьютерами. **Важно соблюдение табуляции(!!!)**

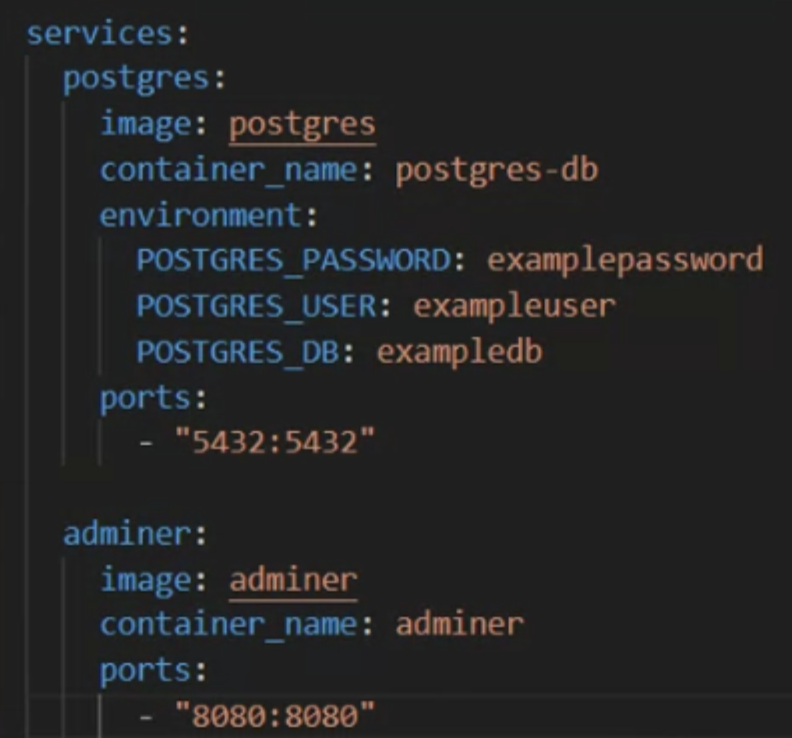
Все необходимые контейнеры запускаются и выключаются одной командой (docker-compose up u docker-compose down);

Автоматическое создание необходимых образов на основании Dockerfile каждого приложения;

Автоматическое создание изолированной сети для взаимодействия контейнеров;

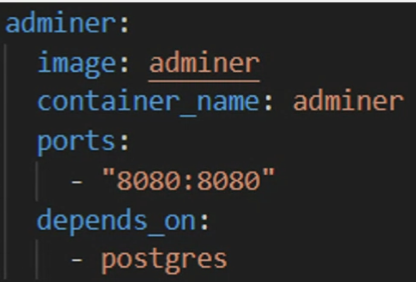
С помощью DNS возможно взаимодействие между контейнерами, используя им.

**services** определяются все контейнеры, которые вы хотите запустить с помощью docker-compose. Каждый сервис имеет имя и определенные настройки.

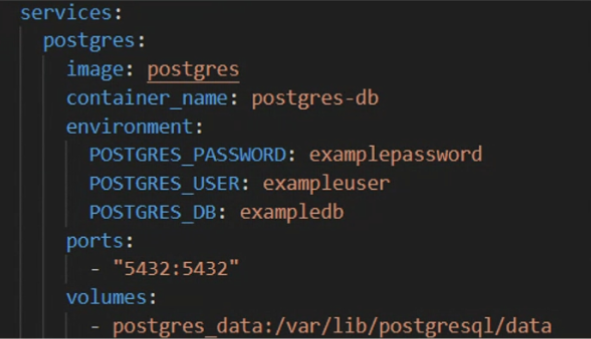


Можно настроить различные параметры для каждого сервиса, такие как переменные окружения, порты, названия контейнера и тд. Переменные окружения в Docker Compose позволяют определить и передать конфигурационные параметры в контейнеры.

**depends\_on** Можно указать зависимости между сервисами, чтобы гарантировать, что один сервис запускается только после запуска другого.

****

**volumes** Можно указать тома, чтобы сохранить данные между запусками контейнеров. Все, что находиться в volumes сохраняется на диске после удаления контейнера. Остальные данные хранятся в памяти.



**docker-compose ps** Отображение статуса контейнеров, запущенных с помощью Docker Compose.

**docker-compose logs** Просмотр логов контейнеров.

**docker-compose build** Построение образов, определенных в docker-compose.yml.

**docker-compose exec** < service-name> <command > Выполнение команды внутри контейнера.

# ПРАКТИЧЕСКАЯ РАБОТА №1

Работа с файлами в локальном репозитории Git.

1. Настройка конфигурации и вывод конфигурации Git.

При первом создании локального Git-репозитория с использованием GitBash необходимо установить базовую конфигурацию – настроить пользователя и электронный адрес.

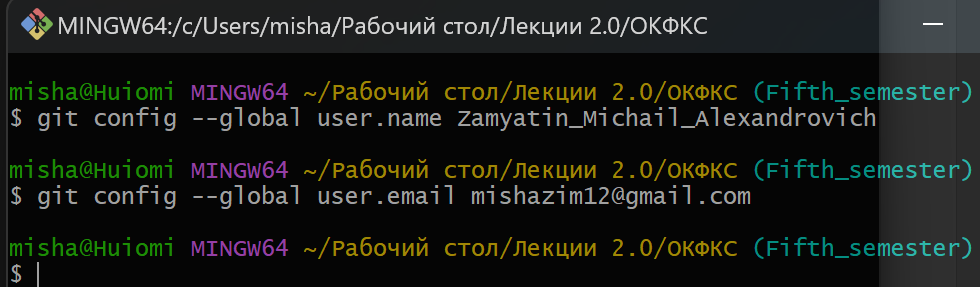


Рисунок 1 - Настройка пользователя.

Следующая команда настроит русификатор – файлы с латиницей в названии будут корректно отображаться (git config –global core.quotePath false).

Для того, чтобы отобразить текущую настройку конфигурации Git необходимо ввести команду «git config --list».

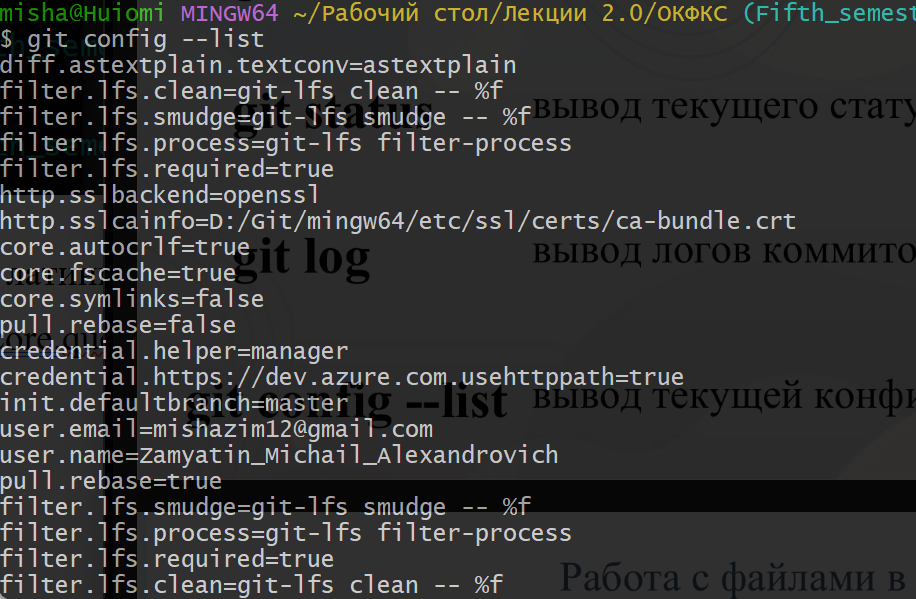


Рисунок 2 - Лист конфигурации.

2. Создание локального репозитория.

Для того, чтобы создать локальный репозиторий необходимо открыть консоль/PowerShell/GitBash в нужной папке и выполнить следующую команду.

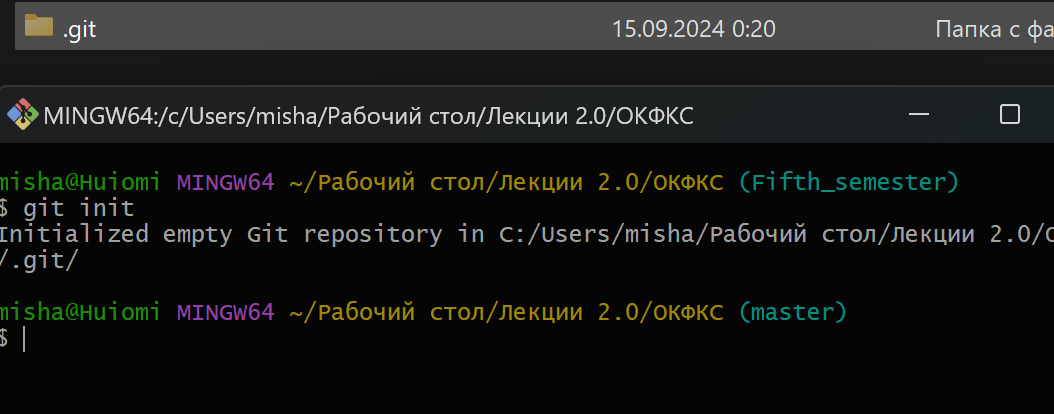


Рисунок 3 - Создание репозитория.

Для удаления репозитория можно использовать команду «rm -rf .git».

3. Работа с файлами в репозитории.

Для того, чтобы добавить все файлы из папки, в которой расположен наш локальный репозиторий нужно прописать команду «git add .». Для добавления конкретного файла – «git add file\_name».

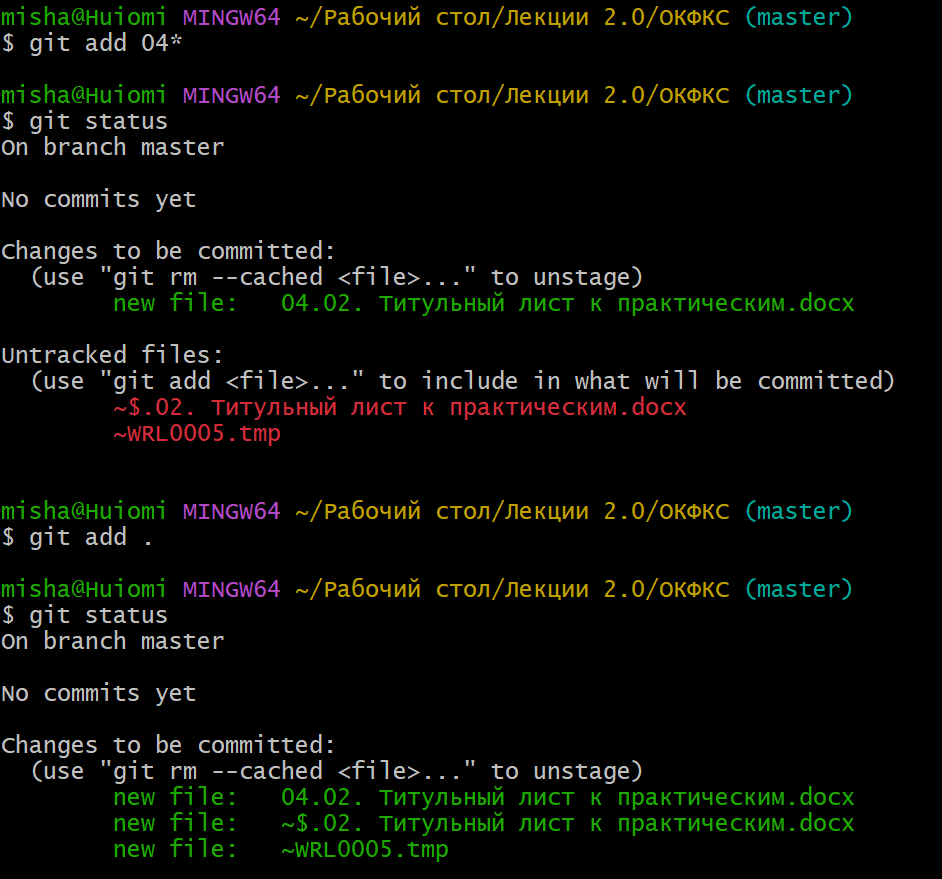


Рисунок 4 - Добавление файлов.

Команда «git status» выводит состояние репозитория. Чтобы удалить файл, нужно ввести команду «git rm –cached file\_name». Однако, при использовании команды «git rm -f file\_name» можно удалить файл с устройства (переместить в корзину).

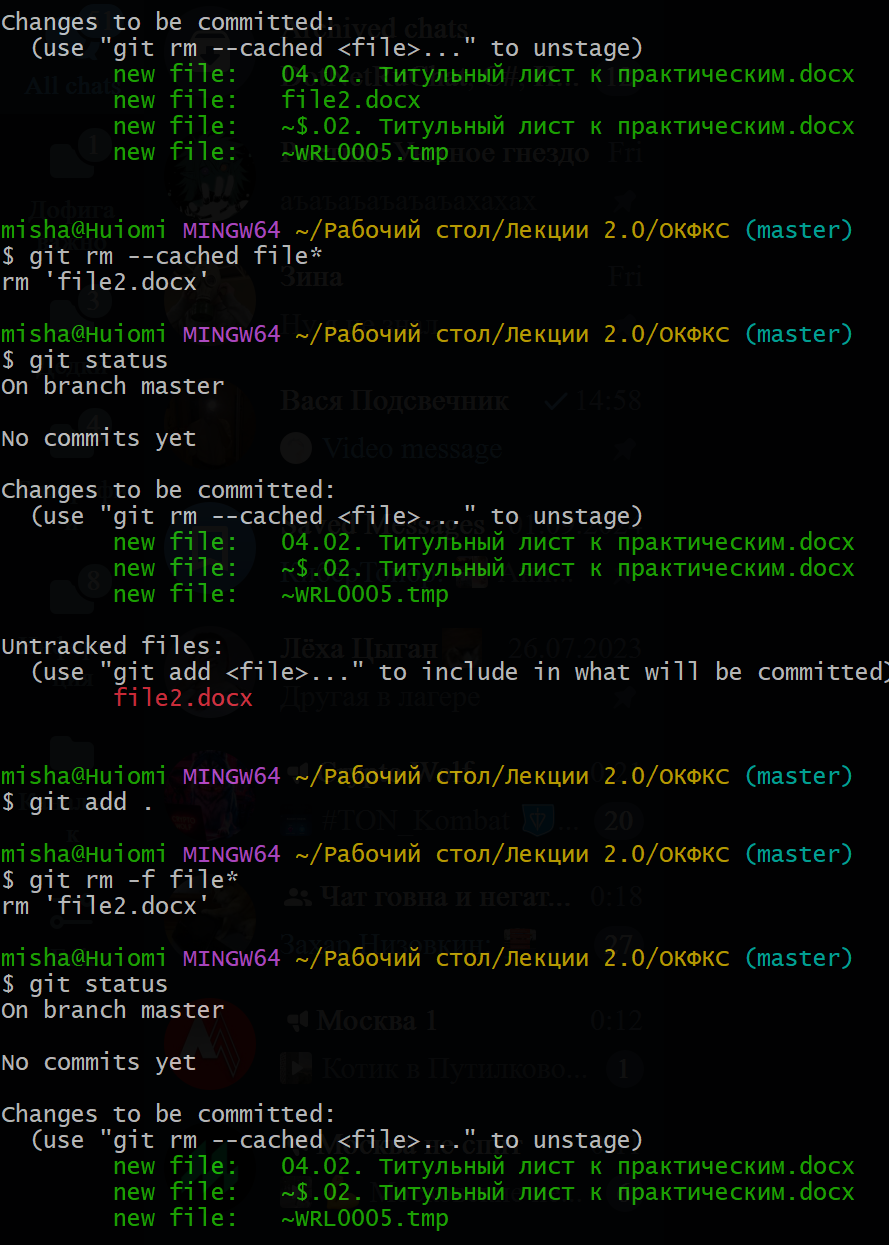


Рисунок 5 - Удаление файлов.

Так же мы можем отматывать файл на последнюю зафиксированную версию – «git restore file».

4. Работа с указателями «HEAD».

Для создания коммита используется команда «git commit -m “commit\_name”».

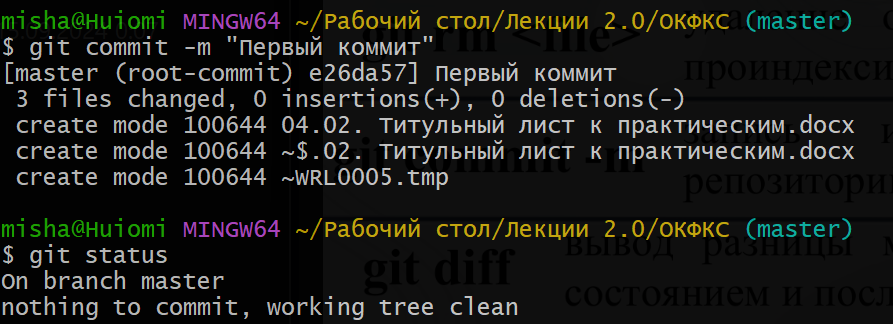


Рисунок 6 - Создание коммита.

После создания коммита у нас появляется указатель «HEAD», который указывает на последний коммит в текущей ветке. Его можно смещать – HEAD~ или HEAD^.

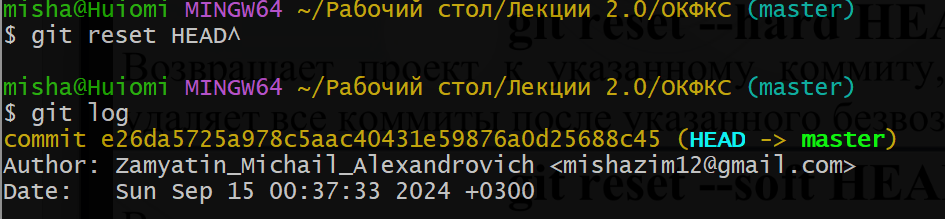


Рисунок 7 - Откат назад по коммитам.

Чтобы вывести информацию по отслеживаемому репозиторию и по предыдущим, вводиться команды «git log».

Помимо прямого перемещения назад по коммитам с удалением всех внесенных изменений можно перемещаться при помощи других команд: «git reset –hard commit\_name» и «git reset –soft commit\_name».

При использовании первой команды новые файлы удаляются, а старые восстанавливаются. Все изменения в файлах тоже восстанавливаются на уровень названного коммита.

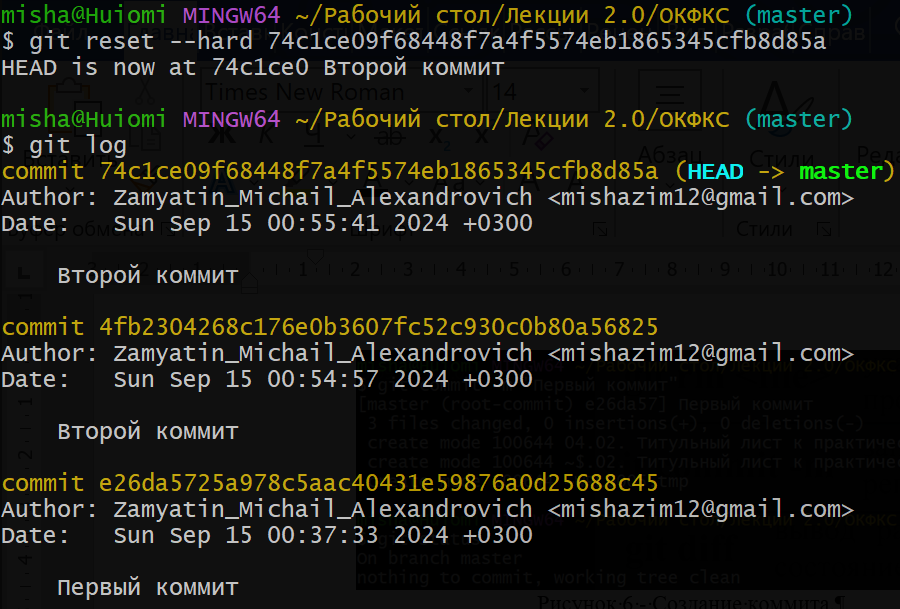


Рисунок 8 - Переход к определенному коммиту.

При использовании второй команды старые файлы остаются на уровне последнего коммита, но происходит переход на другую ветку.

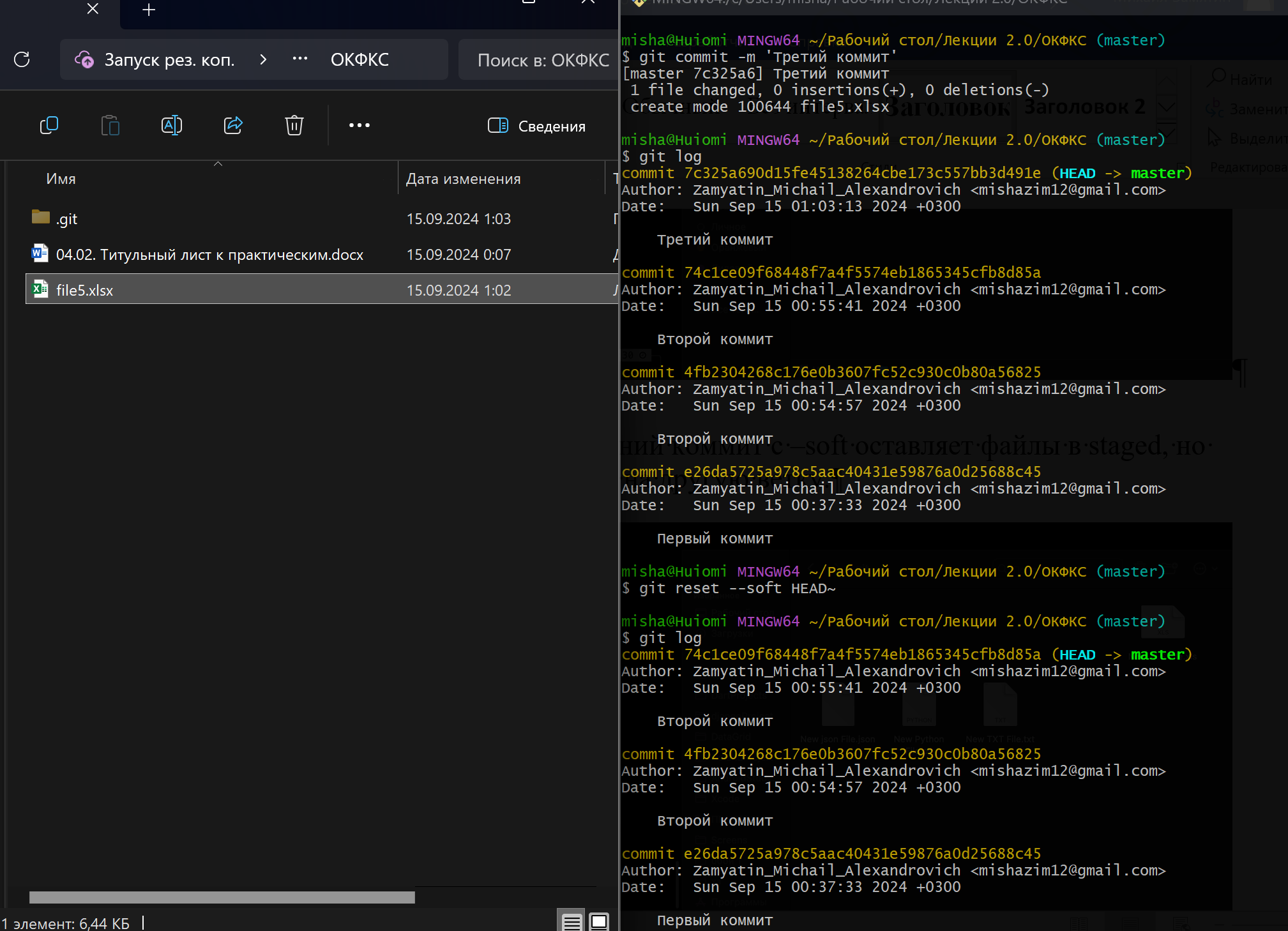


Рисунок 9 - Soft переход.

Вывод: проведена ознакомительная работа с локальным репозиторием Git, а именно: настройка конфигурации, операции над файлами, создание коммитов и демонстрация работы откат-системы в них.

# ПРАКТИЧЕСКАЯ РАБОТА №2

Работа с удаленным репозиторием и веткой.

Цель работы: научиться работать с удаленным репозиторием и ветками на нем, работа с командами pull, merge, fetch.

Для работы с ветками в необходимо использовать команду «git branch». Она позволяет посмотреть, какая ветка сейчас активная, а также создать новую

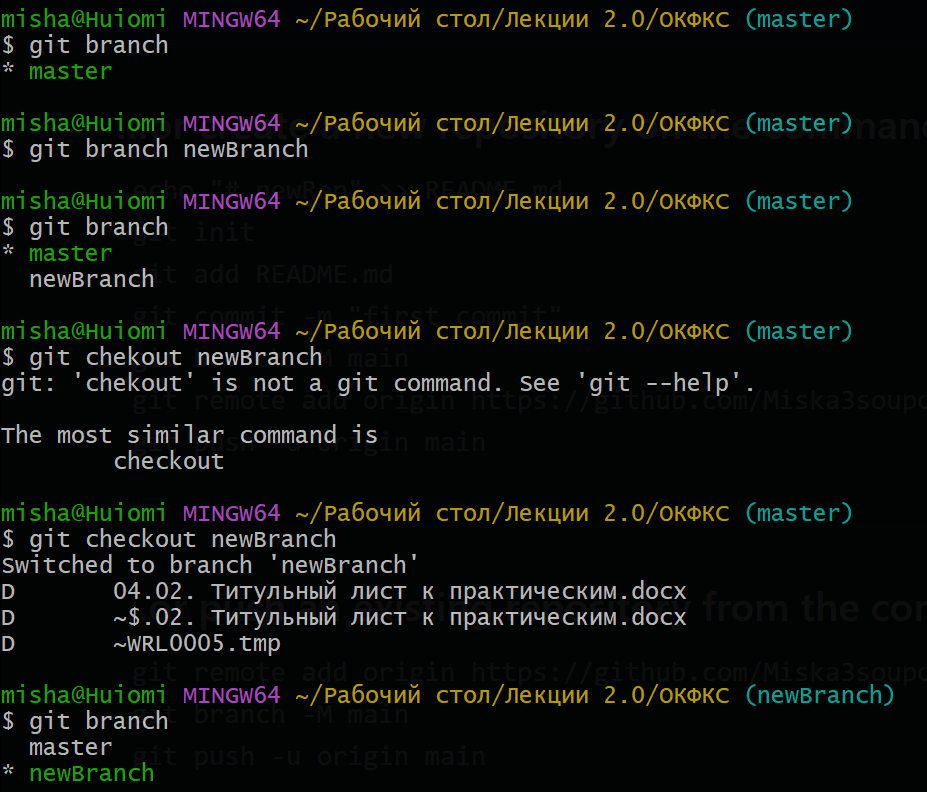


Рисунок 10 - Создание ветки.

Можно заметить, что все файлы на ветке master сохранили своё состояние. Однако они не перенеслись на новую ветку. Это создано для удобной работы с изменениями файлов.

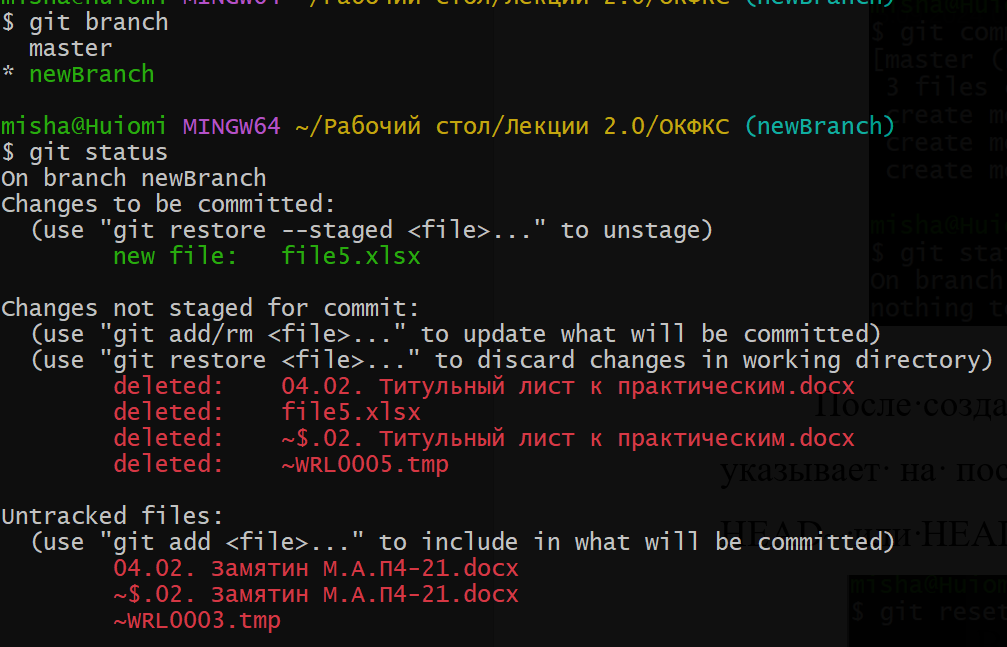


Рисунок 11 - Файлы в ветке.

На GitHub есть несколько вкладок, которые постоянно используются при работе с данной web-сервисом.

При входе на сайт пользователя встречает главная страница, где он может посмотреть список подписчиков («followers») и подписок («following»), а также часто используемые репозитории.

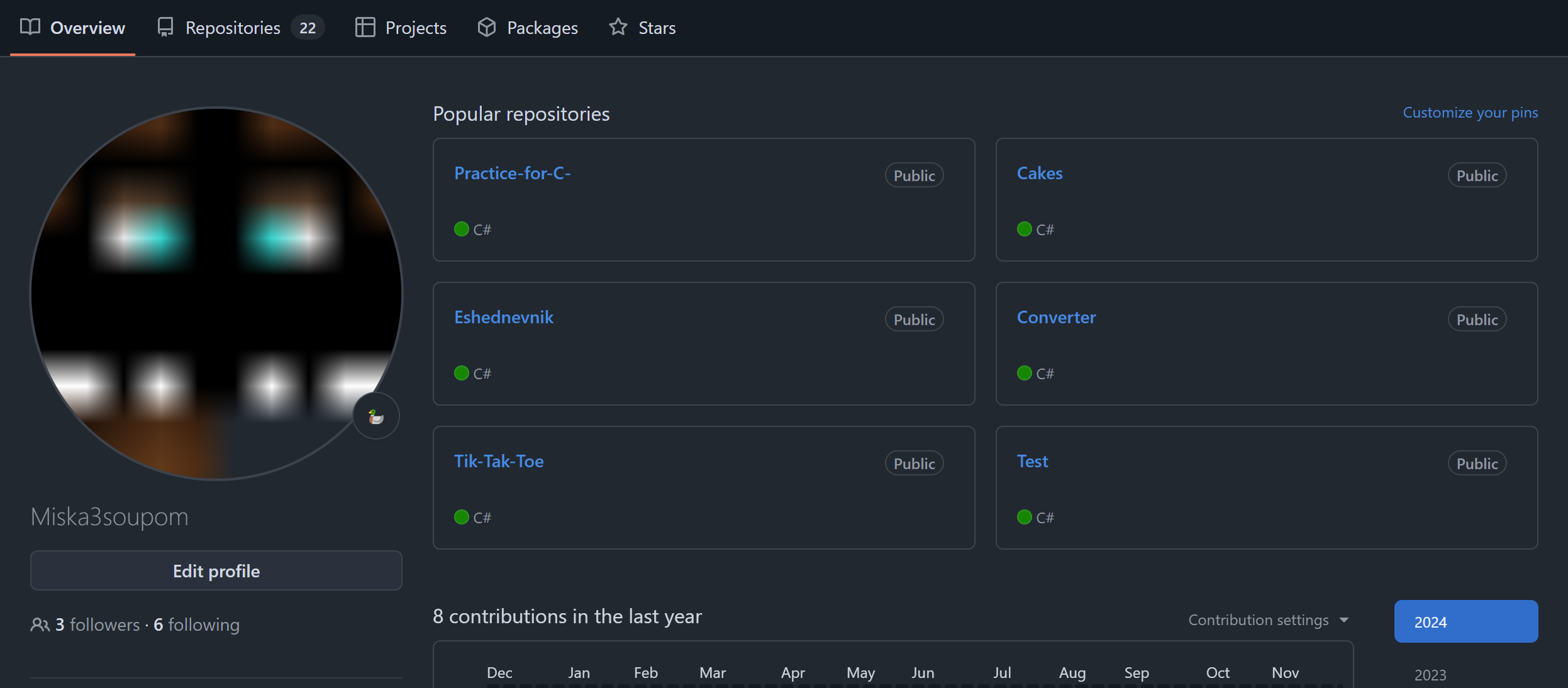


Рисунок 12 - Главная страница.

Во вкладке «Repositories» находятся все репозитории пользователя, а также организации и совместные репозитории, к которым он допущен или приглашен.

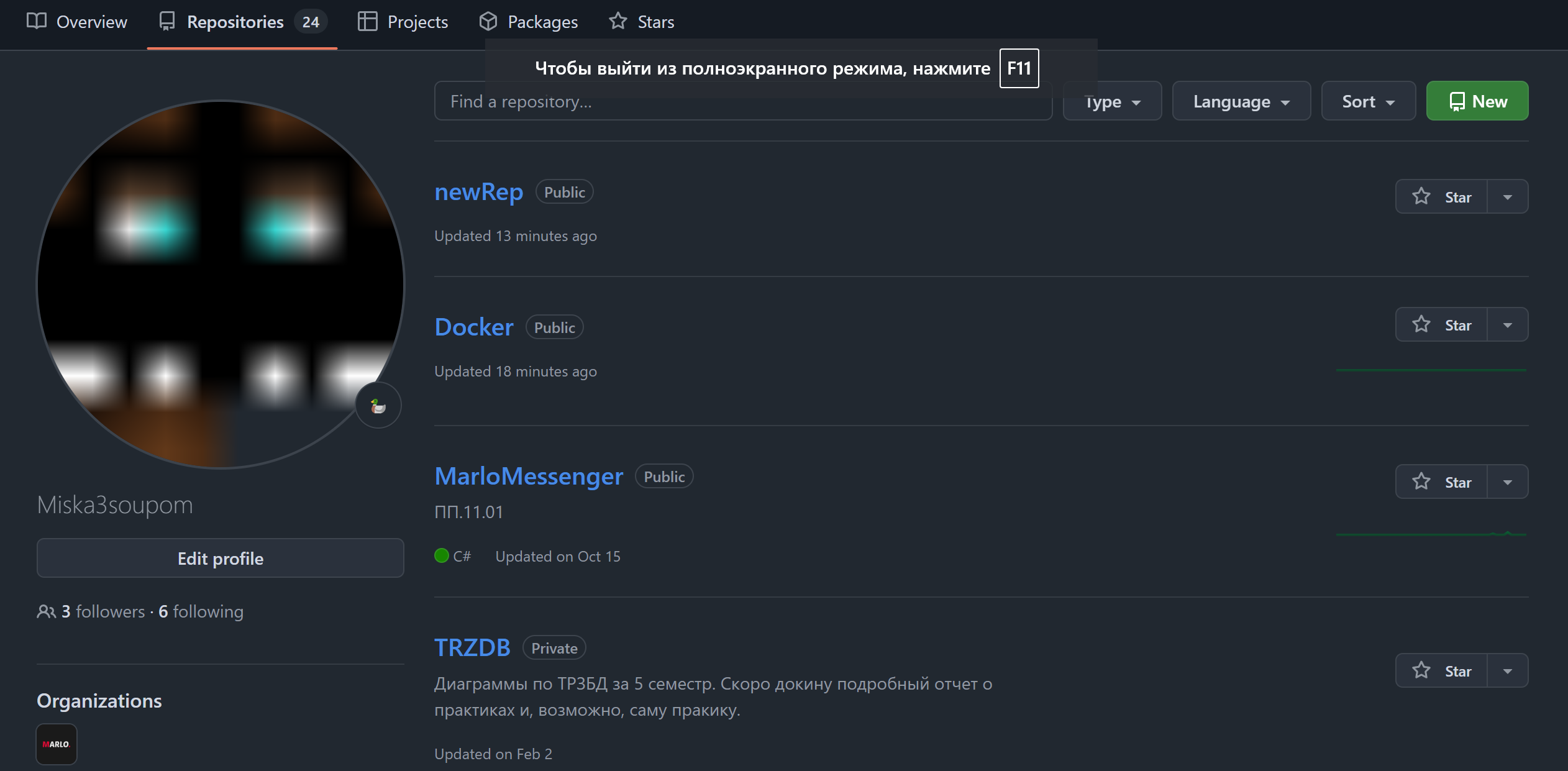


Рисунок 13 - Репозитории.

При нажатии на кнопку «New» можно создать новый репозиторий. При переходе во вкладку «Projects» пользователю показывает его рабочие проекты. Вкладка «Packages» отображает дополнительные пакеты, которые можно использовать совместно с GitHub.

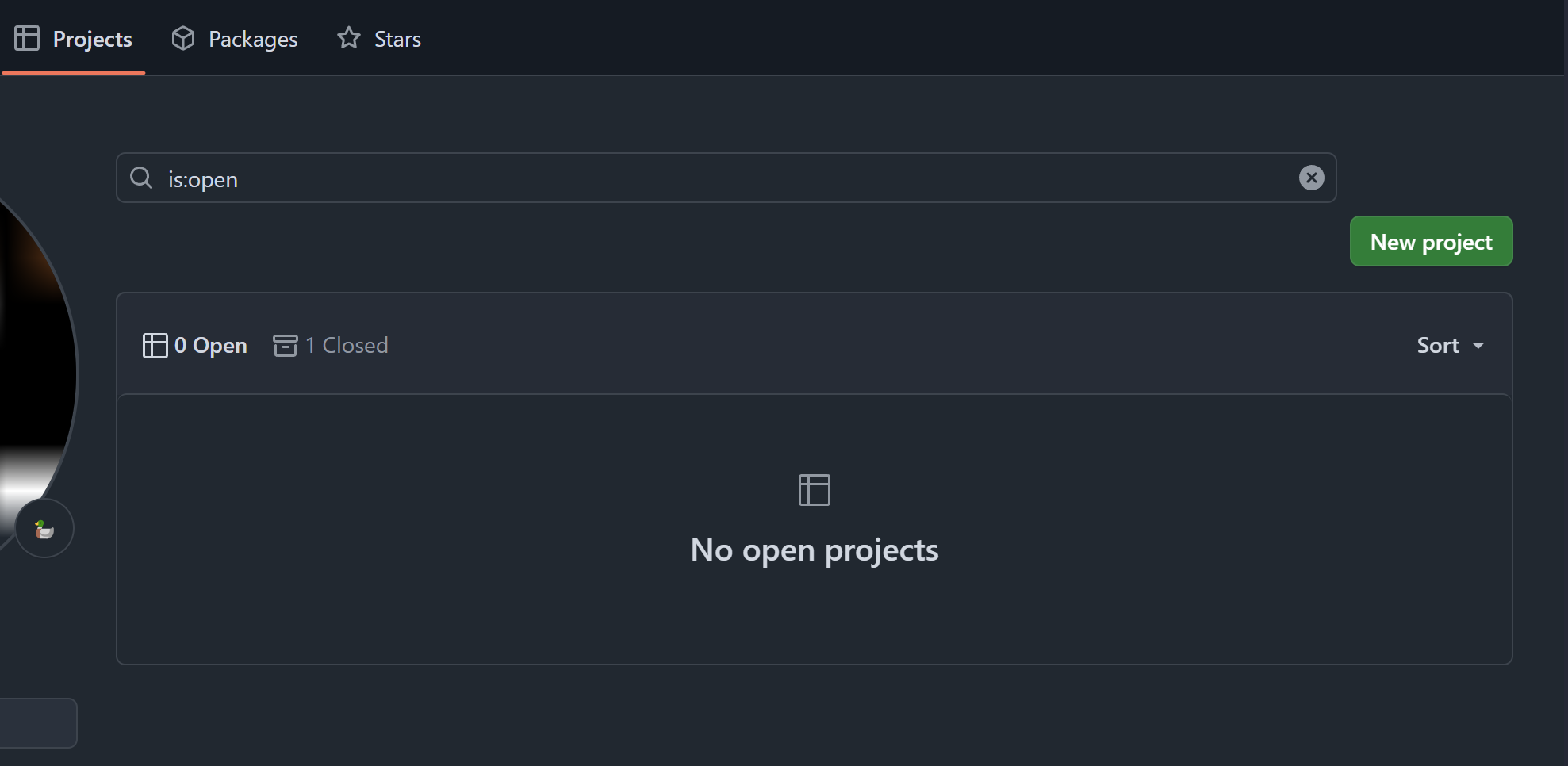


Рисунок 14 - Проекты.



Рисунок 15 - Пакеты.

Создадим удаленный репозиторий в GitHub.

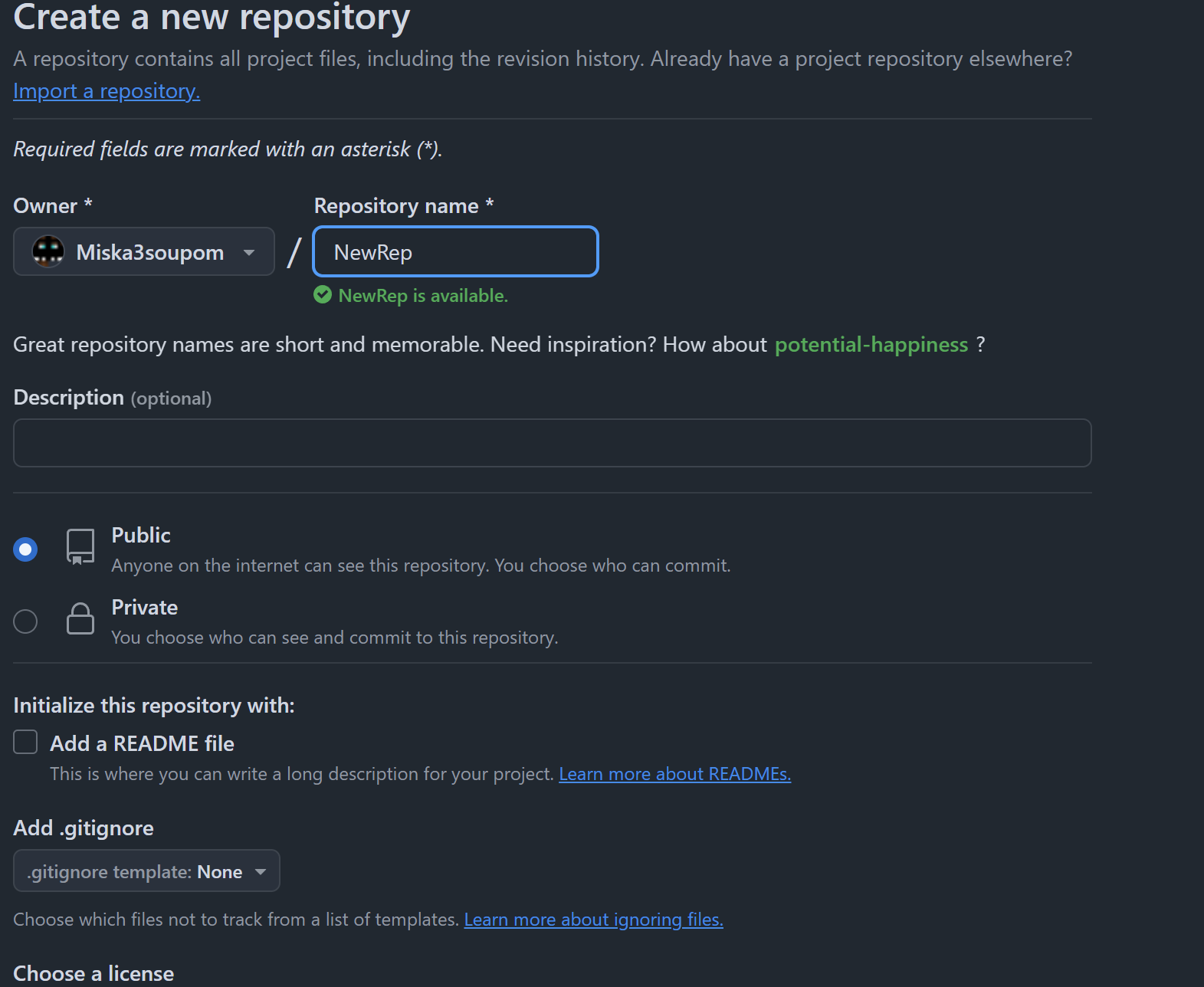


Рисунок 16 - Создание нового репозитория.

Создадим папку и проинициализируем новый репозиторий. При помощи команды «git push» добавим наши файлы из локального репозитория на удаленный и с помощью команды clone создадим локальную копию репозитория.

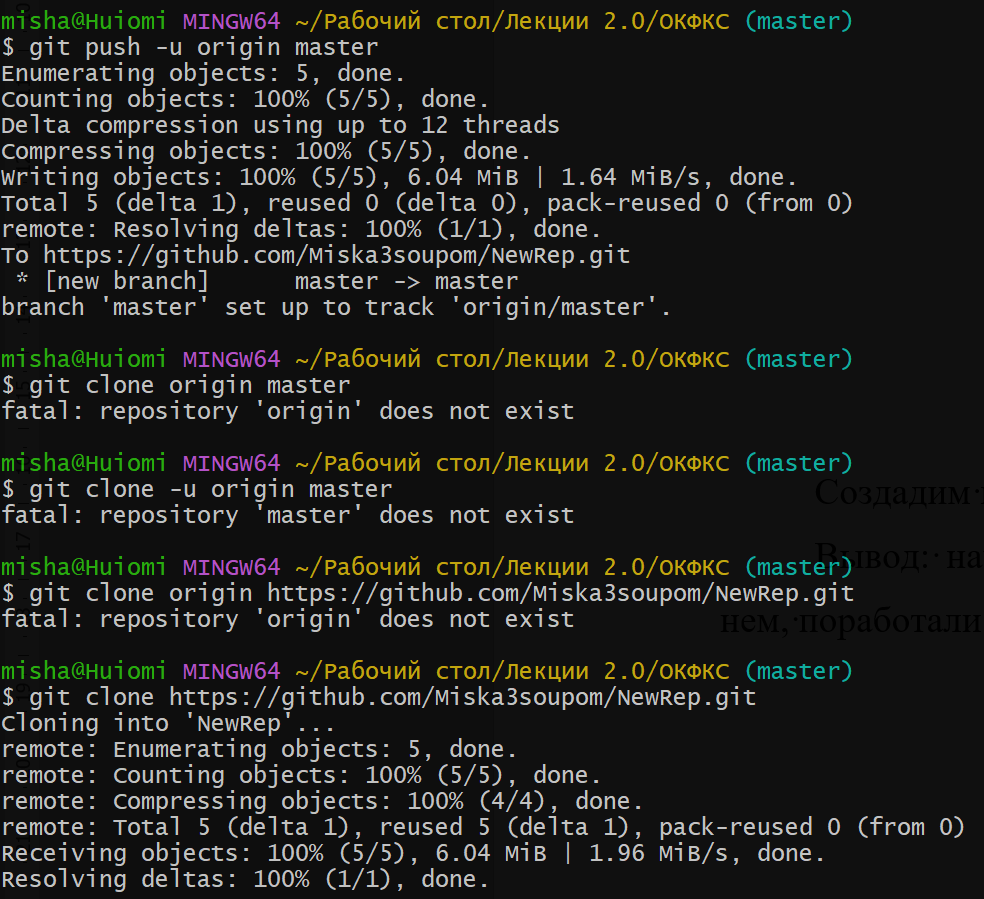


Рисунок 17 - push и сlone.

Вывод: научились работать с удаленным репозиторием и ветками на нем, поработали с командами pull, merge, fetch.