|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных технологий

**Отчет по практической работе №1**

по дисциплине «Технологии разработки программных приложений»

**Тема практической работы:** «Системы контроля версий»

|  |  |
| --- | --- |
| **Выполнил:**  Студент группыИКБО-26-21 | Оганнисян Г.А. |
| **Проверил:** | кандидат технических наук, доцент Жматов Д.В. |

Москва 2023

**Цель практической работы**

Целью данной практической работы является получение навыков по работе с командной строкой и git’ом.

**Выполнение практической работы**

**Часть 1. Как начать работу с git’ом?**

1. **Установка git на локальную машину**

Для начала скачаем exe-файл инсталлятора с сайта git (<https://git-scm.com/download/win>).

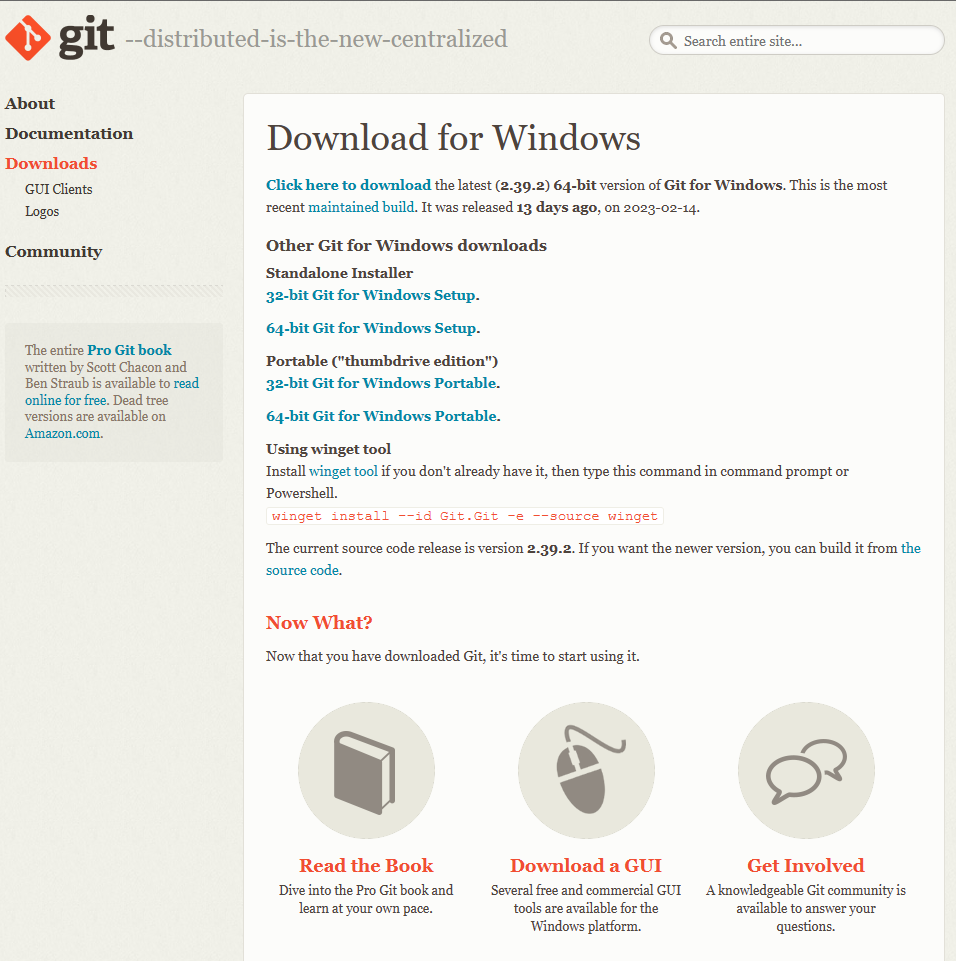


Рисунок 1.1 – Сайт git

Теперь запустим установщик.



Рисунок 1.2 – Запущенный установщик git

1. **Настройка Git**

Откроем терминал и выполним две команды.

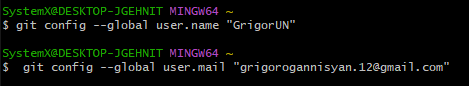


Рисунок 2.1 – Команды для задания имени и электронной почты

Далее выполним следующие команды: git config --global core.autocrlf true, git config --global core.safecrlf warn и git config --global core.quotepath off. После их выполнения проверим правильность введённых данных при помощи команды git config –list.

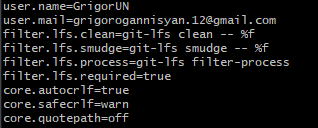


Рисунок 2.2 – Проверка правильности введённых данных

1. **Начало работы – создание папок и файлов**

Для перехода в другую директорию воспользуемся командой cd <путь к директории>. Перейдём в директорию Documents.



Рисунок 3.1 – Переход в директорию Documents

При помощи команды mkdir создадим папку с названием repozitorii и перейдём в неё при помощи команды cd.

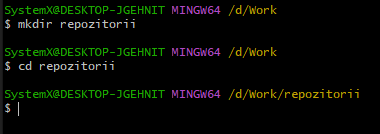


Рисунок 3.2 – Создание папки repositorii

При помощи команды touch создадим новый файл с названием proekt и расширением html.



Рисунок 3.3 – Создание файла proekt.html

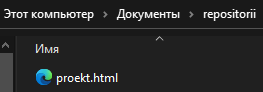


Рисунок 3.4 – Проверка наличия созданного файла

Воспользуемся командой nano для добавления содержания в файл.

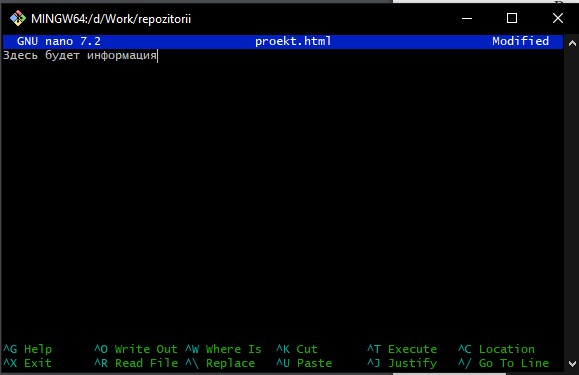


Рисунок 3.5 – Переход в файл командой nano

1. **Создание репозитория**

Для создания репозитория выполним команду git init.



Рисунок 4.1 – Создание нового репозитория

1. **Добавление файла в репозиторий и добавление первого коммита**

Чтобы добавить файл в репозиторий необходимо выполнить команды:

git add <Название файла> и git commit -m "Ваш текст для коммита".

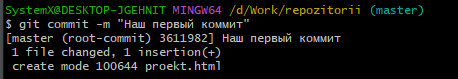


Рисунок 5.1 – Добавление коммита

Выполним команду git status.



Рисунок 5.2 – Выполнение команды git status

При помощи команды nano внесём небольшое изменение в файл и снова запустим команду git status.

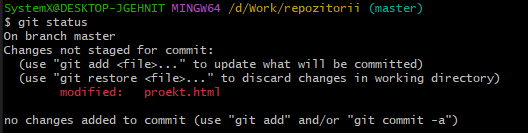


Рисунок 5.3 – Проверка внесения изменений в файл

1. **Индексация изменений**

Выполним команды git add proekt.html и git status.

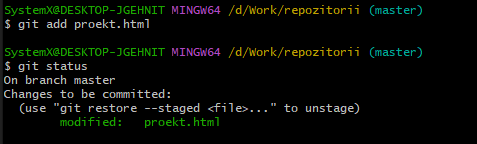


Рисунок 6.1 – Индексация изменений

1. **Коммиты нескольких изменений**

При помощи команды nano внесём изменения в файл.

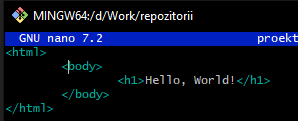


Рисунок 7.1 – Внесение изменений в файл

Теперь снова выполним команду git add «Название файла» и внесём второе изменение. После этого, выполним команду git status.

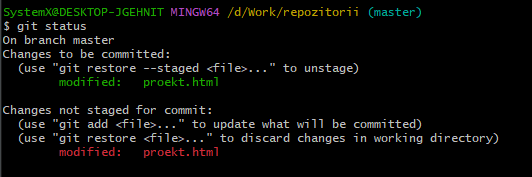


Рисунок 7.2 – Результат выполнения команды git status

Добавим коммит и проверим статус репозитория.

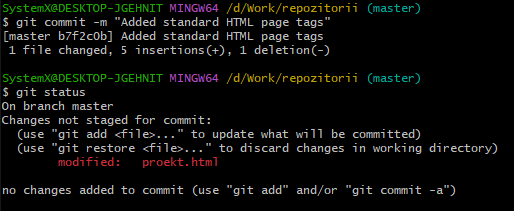


Рисунок 7.3 – Добавление коммита и проверка статуса репозитория

Выполним команды git add . и git status.

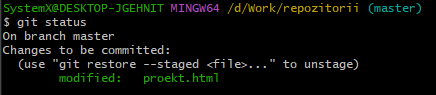


Рисунок 7.4 – Результат выполнения двух команд

Далее делаем коммит второго изменения.

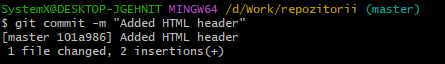


Рисунок 7.5 – Коммит второго изменения

1. **Просмотр истории коммитов**

Для просмотра истории коммитов используется команда git log.

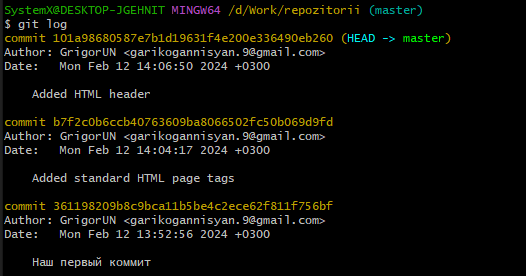


Рисунок 8.1 – Просмотр истории коммитов

Команда log позволяет контролировать формат выводимой информации: git log --pretty=oneline.

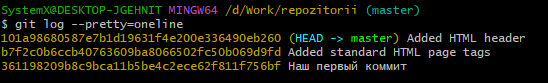


Рисунок 8.2 – Результат выполнения команды git log –pretty=oneline

Также выполним команду git log –pretty=format:«%h %ad | %s%d [%an]» --graph – date=short.

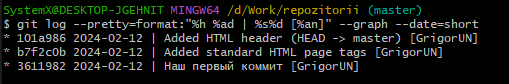


Рисунок 8.3 – Результат выполнения команды

1. **Получение старых версий**

Воспользуемся последней командой из предыдущего пункта, чтобы получить необходимый хэш.

Теперь введём команду git checkout <хэш>.

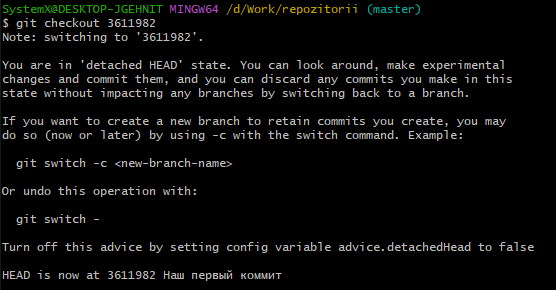


Рисунок 9.1 – Результат выполнения команды git checkout

Проверим наличие данных в файле командой cat proekt.html



Рисунок 9.2 – Проверка данных файла

Вернемся в нынешнее последнее состояние при помощи команды git checkout и проверим, что находится в файле с помощью команды cat.

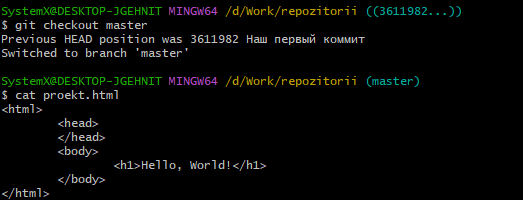


Рисунок 9.3 – Возврат в текущее состояние файла

1. **Отмена локальных изменений (до индексации)**

Убедимся, что мы находимся на последнем коммите ветки master, прежде чем продолжить работу, для этого введем команду git checkout master.



Рисунок 10.1 – Проверка нахождения на последнем коммите ветки master

Изменим наш файл в рабочем каталоге при помощи команды nano и сохраним его.

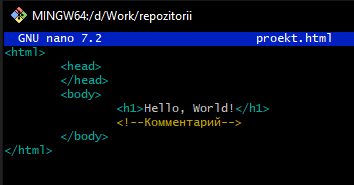


Рисунок 10.2 – Внесение изменений в файл

Проверим состояние рабочего каталога при помощи команды git status.

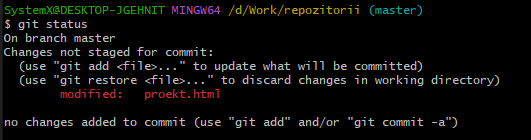


Рисунок 10.3 – Проверка состояния рабочего каталога

Для отмены изменений воспользуемся следующими командами и проверим, что изменилось.



Рисунок 10.4 - Результат выполнения команды git checkout

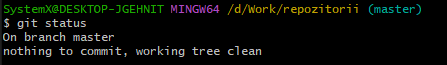


Рисунок 10.5 – Результат выполнения команды git status

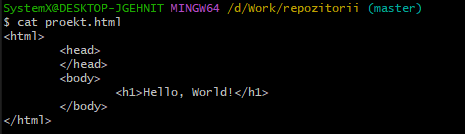


Рисунок 10.6 – Проверка отмены локальных изменений

1. **Отмена локальных изменений (после индексации и до коммита)**

Снова внесём изменения в файл.

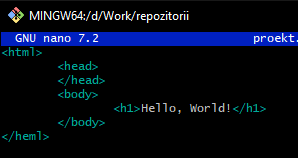


Рисунок 11.1 – Внесение изменений в файл

Проиндексируем наши изменения с помощью команды git add и проверим состояние нашего изменения при помощи команды git status.

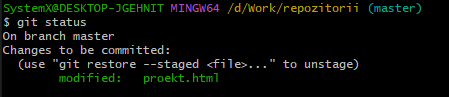


Рисунок 11.2 – Проверка состояния нашего изменения

Отменим индексацию изменений.

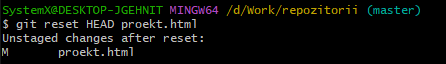


Рисунок 11.3 – Отмена индексации изменений

Уберём ненужный комментарий при помощи команды git checkout.



Рисунок 11.4 – Удаление ненужного комментария

Выполним команду git status для проверки статуса рабочего каталога.

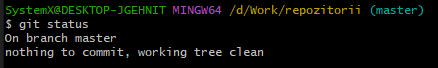


Рисунок 11.5 – Результат проверки статуса рабочего каталога

1. **Отмена коммита**

Внесем снова изменения в наш файл при помощи команды nano.

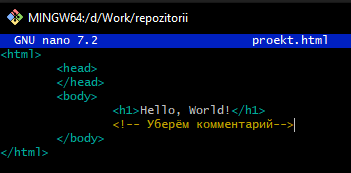


Рисунок 12.1 – Внесение изменений в файл

Проиндексируем наше изменение при помощи команды git add.

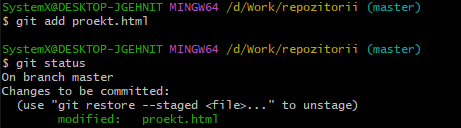


Рисунок 12.2 – Индексация изменения

Произведем следующий коммит при помощи команды git commit –m.

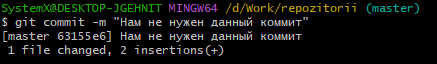


Рисунок 12.3 – Добавление коммита

Чтобы отменить коммит, нам необходимо сделать коммит, который удаляет изменения, сохраненные нежелательным коммитом, для это воспользуемся следующей командой git revert HEAD –no-edit.

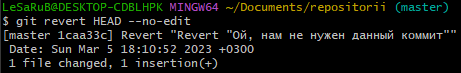


Рисунок 12.4 – Отмена коммита

Проверим историю коммитов.

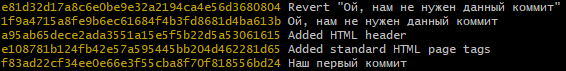


Рисунок 12.5 – Проверка истории коммитов

**Часть 2. Управление репозиториями**

Так как аккаунт на Git Hub уже был создан ранее, шаг с регистрацией пропускаем.

1. **Создание SSH-ключа для авторизации**

Перейдём в каталог, хранящий по умолчанию ssh-ключи.

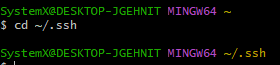


Рисунок 1.1 – Переход в каталог с ssh-ключами

Теперь введём команду ssh-keygen -t rsa -b 4096 –C.

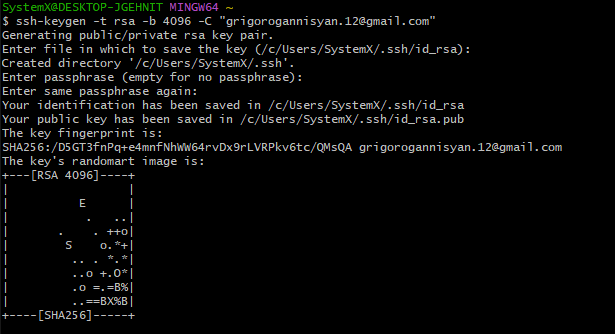


Рисунок 1.2 – Генерация и сохранение ssh-ключа

Добавляем ключ в shh-agent.

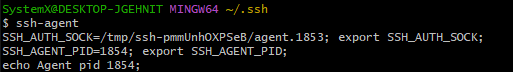


Рисунок 1.3 – Добавление ключа в ssh-agent

Проверяем доступность ключа командой eval "$(ssh-agent -s)".



Рисунок 1.4 – Проверка доступности ssh-ключа

Добавляем с помощью ssh-add ~/.ssh/your\_key\_name, где указываем верный путь до файла с ключом и его имя.



Рисунок 1.5 – Добавление имени ключа

Чтобы связать свой локальный и удаленный репозиторий, выведём в консоль содержимое файла с помощью команды cat.

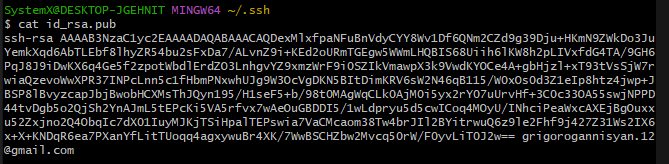


Рисунок 1.6 – Просмотр содержимого файла с ssh-ключом

Скопируем ssh-ключ из консоли и перейдём на страницу для работы с ключами.

Выбираем кнопку “New SSH key”, открывается окно с вводом данных, в поле “key” вставляем скопированный ключ, в “Title” вводим любое имя ключа и нажимаем “Add SSH key”.

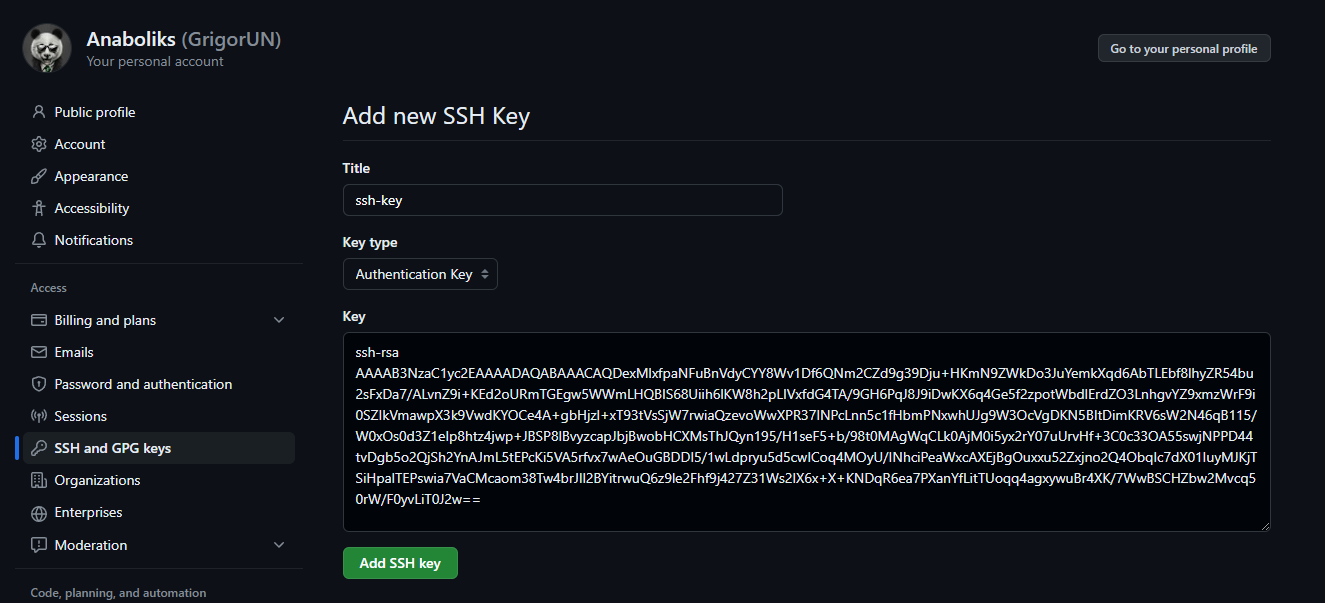


Рисунок 1.7 – Добавление ключа в Git Hub

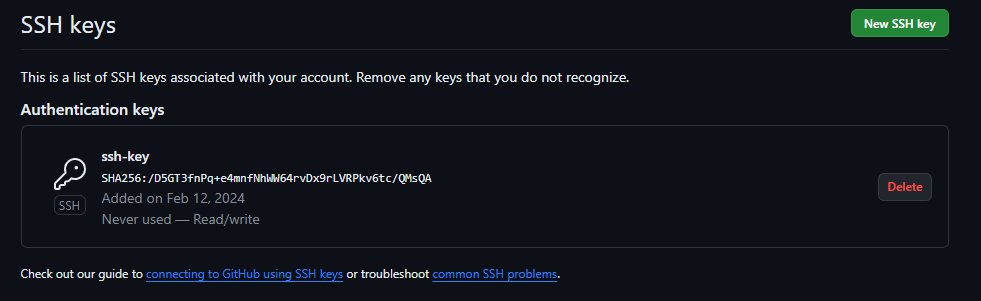


Рисунок 1.8 – Добавленный ключ

1. **Создание нового репозитория для своего проекта**

Создадим новую папку проекта в другой директории.

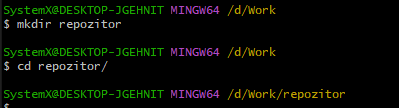


Рисунок 2.1 – Создание новой папки проекта

Теперь добавим несколько файлов.

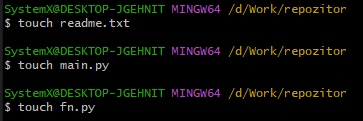


Рисунок 2.2 – Добавление файлов в папку проекта

После добавления файлов создадим новый локальный репозиторий.

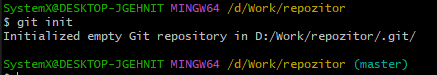


Рисунок 2.3 – Создание нового локального репозитория

1. **Связываем локальный и удалённый репозитории**

Для начала заходим на GitHub, на свою страницу, выбраем вкладку репозитории.

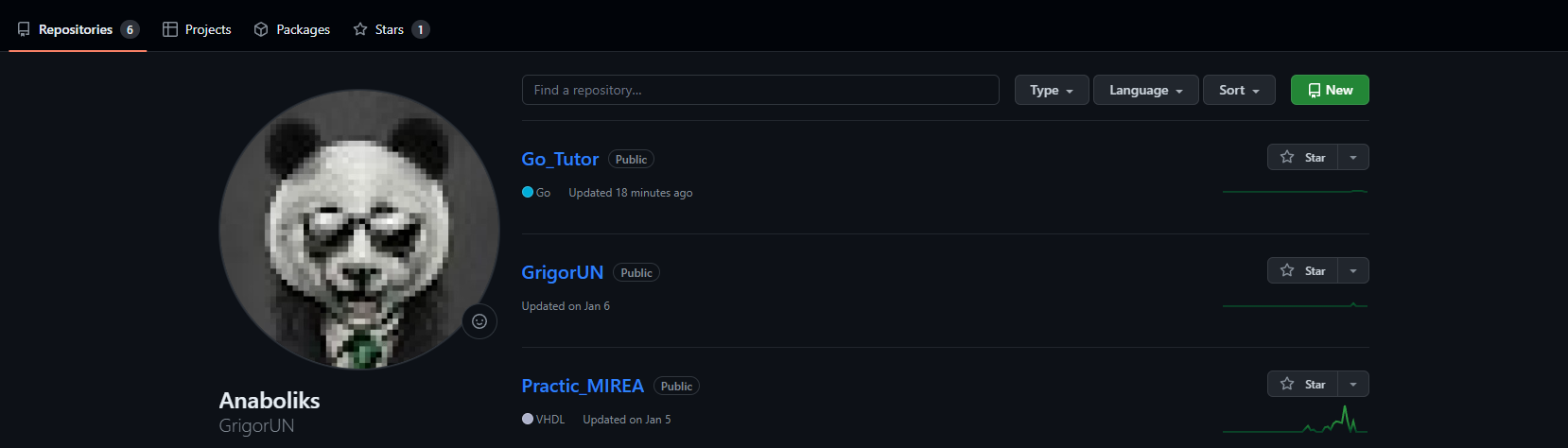


Рисунок 3.1 – Вкладка с репозиториями

Нажмём на кнопку «New». Задаём имя нашему репозиторию и выбираем приватность репозитория.

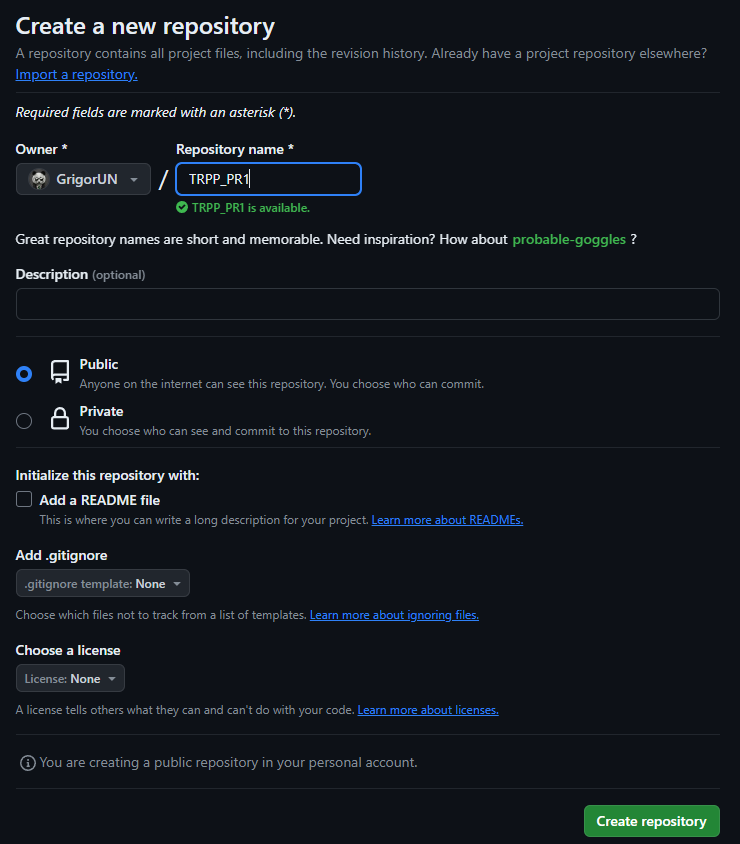


Рисунок 3.2 – Окно создания репозитория

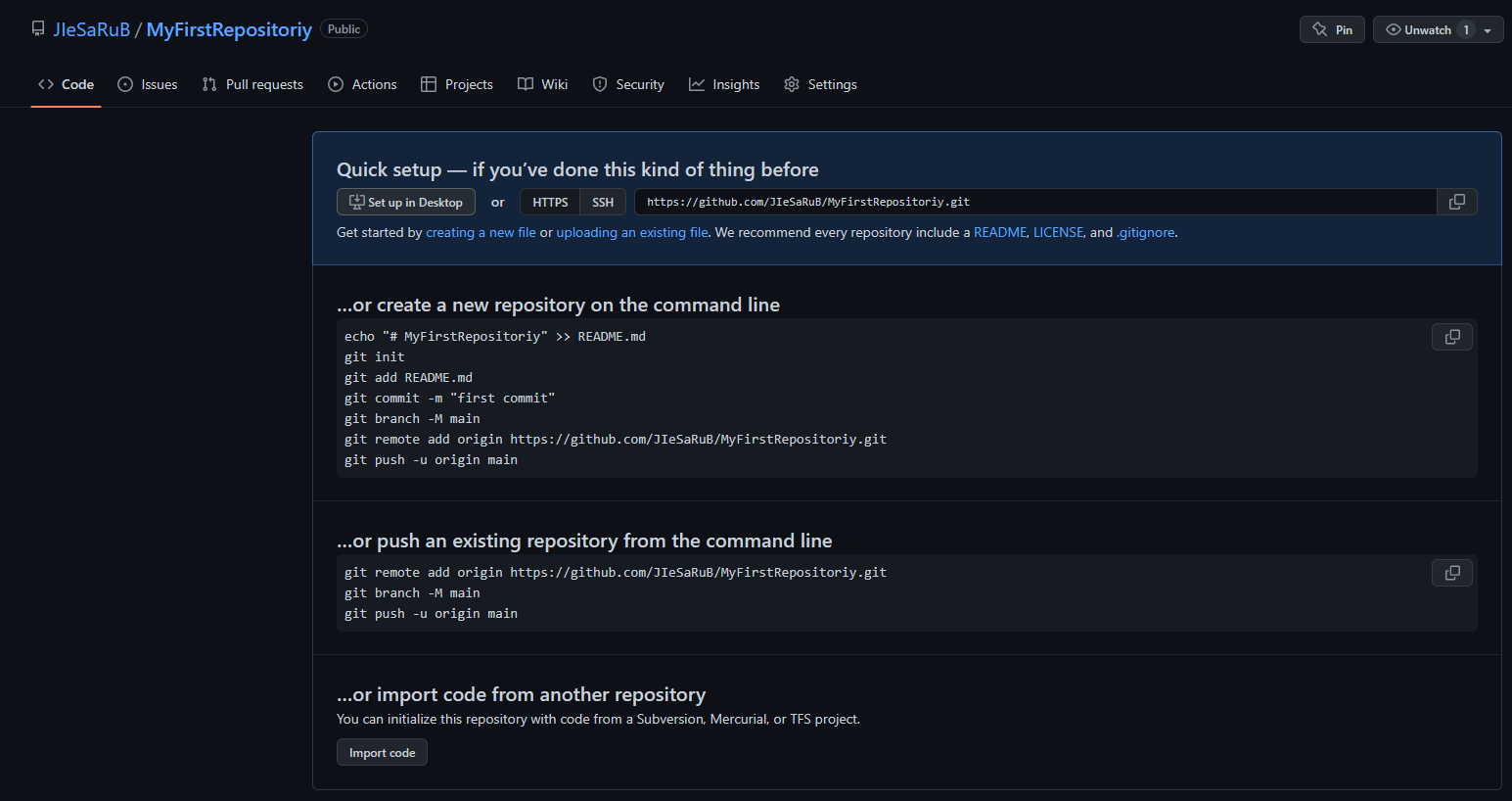


Рисунок 3.3 – Созданный пустой репозиторий

Теперь надо связать созданный на Git Hub репозиторий с нашим локальным репозиторием. Для этого в консоли надо ввести следующую команду.



Рисунок 3.4 – Связывание локального и удалённого репозитория

1. **Создание веток и переключение между ними**

Для создания новой ветки необходимо воспользоваться командой git checkout –b.

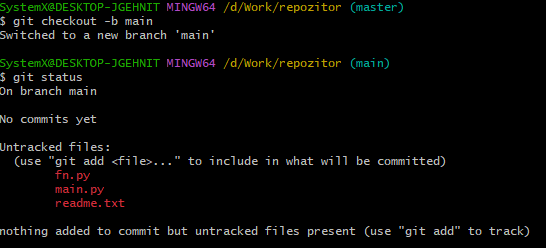
****

Рисунок 4.1 – Создание новой ветки

Теперь внесём некоторые изменения в файлы.

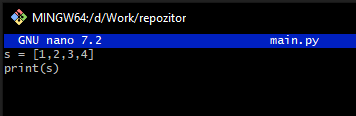


Рисунок 4.2 – Изменения в файле main.py

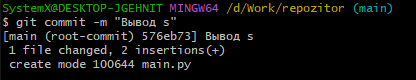


Рисунок 4.3 – Коммит для файла main.py

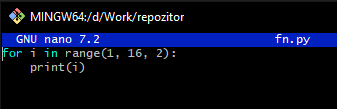


Рисунок 4.4 – Изменения в файле fn.py

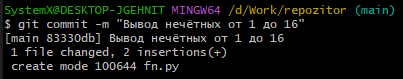


Рисунок 4.5 – Коммит для файла fn.py

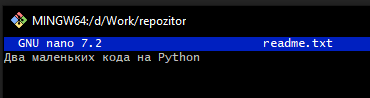


Рисунок 4.6 – Изменение в файле readme.txt

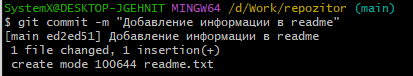


Рисунок 4.7 – Коммит для файла readme.txt

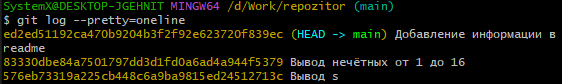


Рисунок 4.8 – Проверка наличия всех коммитов

1. **Слияние веток**

Создадим дополнительную ветку addition.



Рисунок 5.1 – Создание новой ветки



Рисунок 5.2 – Слияние веток

1. **Выполнение индивидуального задания**

**Вариант №9.**

1. Клонируем удалённый репозиторий на локальную машину. Для этого скопируем ссылку на данный репозиторий и выполним команду git clone.

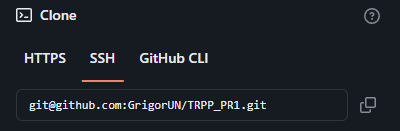


Рисунок 6.1 – Ссылка на удалённый репозиторий

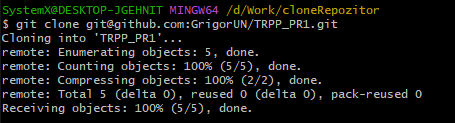


Рисунок 6.2 – Клонирование удалённого репозитория на локальную машину

1. Теперь создадим новую ветку и выведем список всех веток.

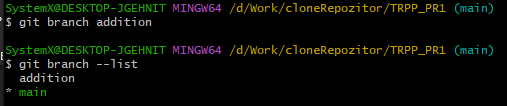


Рисунок 6.3 – Создание новой ветки и вывод списка веток

1. Произведём 5 коммита на разные файлы в новой ветке.

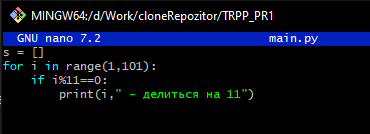


Рисунок 6.4 – Изменение в файле main.py

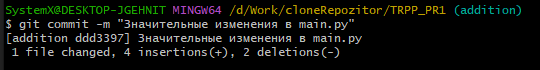


Рисунок 6.5 – Добавление первого коммита

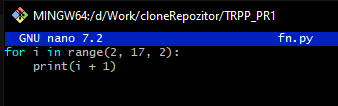


Рисунок 6.6 – Изменение в файле fn.py

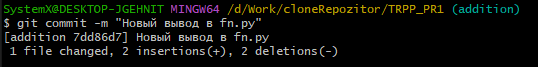


Рисунок 6.7 – Добавление второго коммита

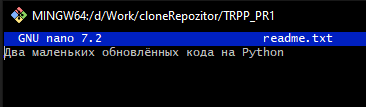


Рисунок 6.8 – Изменение в файле readme.txt

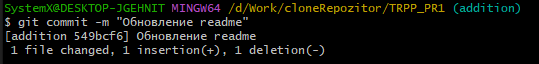


Рисунок 6.9 – Добавление третьего коммита

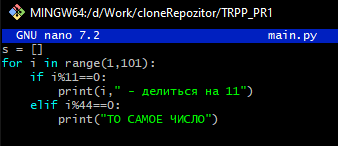


Рисунок 6.10 – Дополнительное изменение в файле main.py

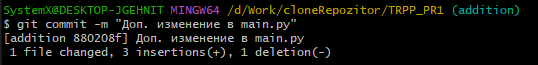


Рисунок 6.11 – Добавление четвёртого коммита

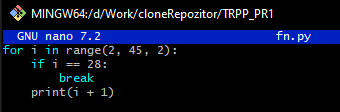


Рисунок 6.12 – Дополнительное изменение в файле fn.py

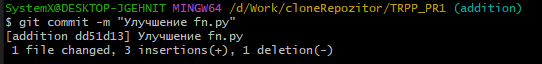


Рисунок 6.13 – Добавление пятого коммита

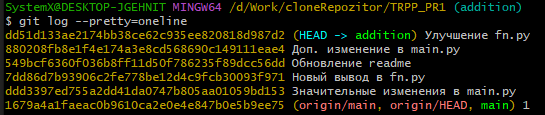


Рисунок 6.14 – Проверка наличия всех коммитов

1. Выгрузим изменения в удалённый репозиторий. Выгрузка в удаленный репозиторий выполняется с помощью команды git push origin.

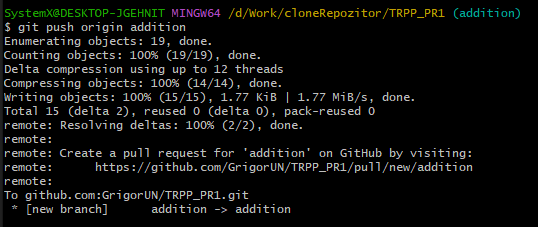


Рисунок 6.15 – Выгрузка изменений в удалённый репозиторий

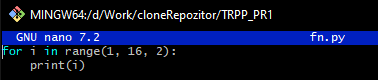
1. Произведём revert 2-го и 4-го коммита в новой ветке. 

Рисунок 6.16 – Revert 2-го коммита.

Проиндексируем изменения и добавим новый коммит

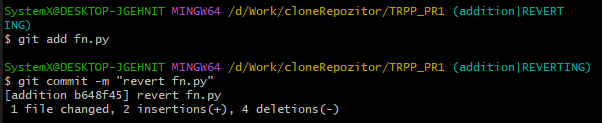


Рисунок 6.17 – Индексация изменений и добавление коммита

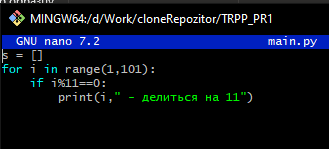


Рисунок 6.18 – Revert 4-го коммита

Выполним команду git revert HEAD и произведём необходимые изменения.

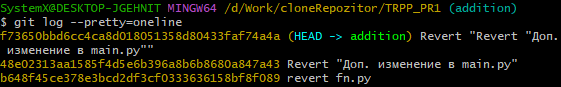


Рисунок 6.19 – Выполнение revert над предпоследним коммитом

1. Выведем в консоли различия между веткой main и новой веткой.

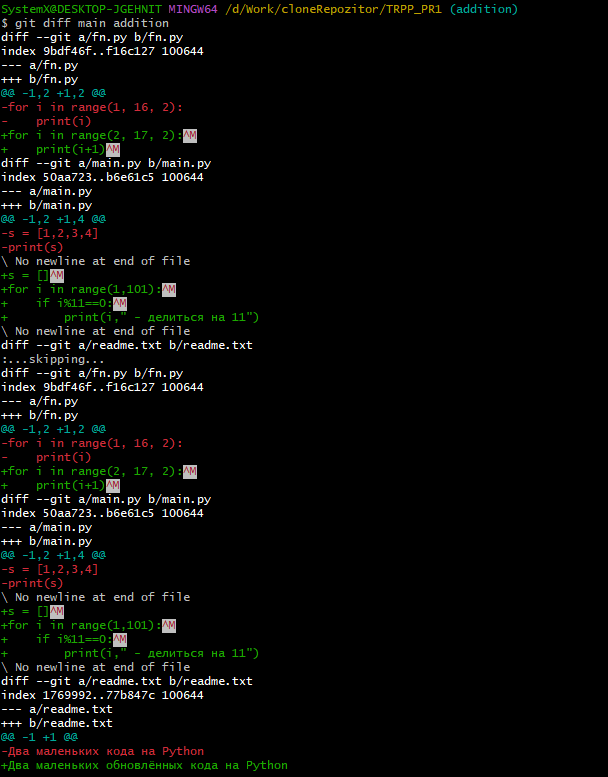


Рисунок 6.15 – Вывод различий между двумя ветками

1. Произведём слияние новой ветки с веткой main при помощи merge.



Рисунок 6.16 – Слияние новой ветки с веткой main

**Часть 3. Выполнение индивидуального задания**

**Вариант №10.**

1. Сделаем форк репозитория в соответствии с вариантом №10. Для этого перейдём по ссылке в необходимый репозиторий и в верхнем правом углу нажмём кнопку Fork.

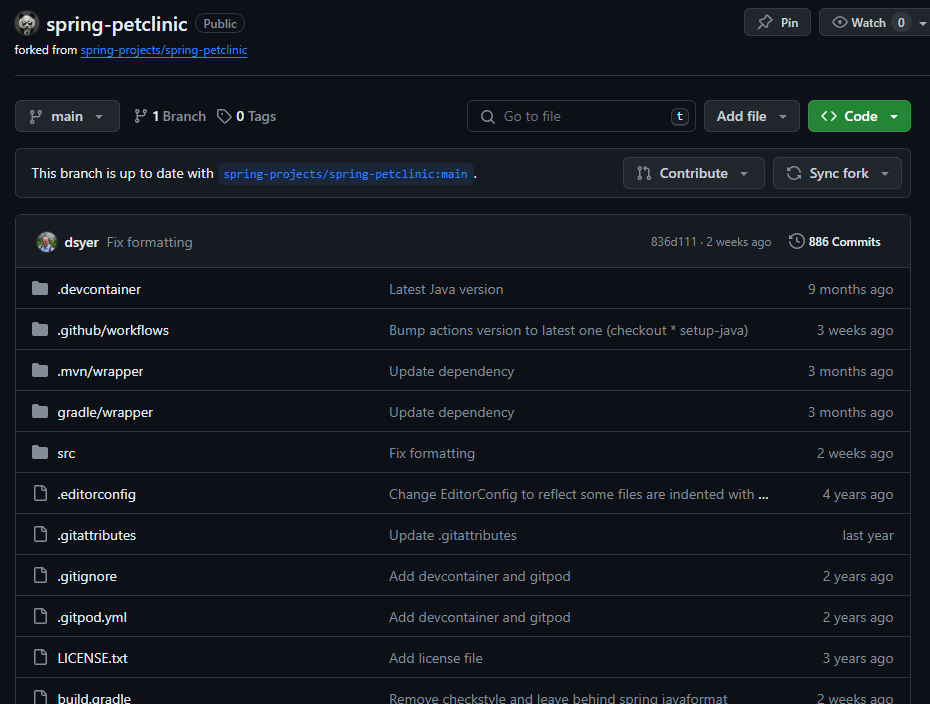


Рисунок 1.1 – Форк необходимого репозитория

1. Склонируем данный репозиторий на локальную машину.

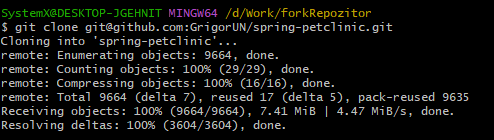


Рисунок 1.2 – Копирование репозитория на локальную машину

1. Создадим две ветки branch1 и branch2.

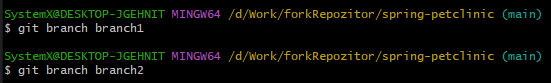


Рисунок 1.3 – Создание новых веток

1. Проведём по три коммита в каждую из веток, которые меняют один и тот же кусочек файла.

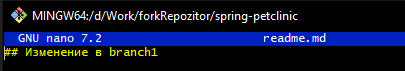


Рисунок 1.4 – Изменение в файле README.md

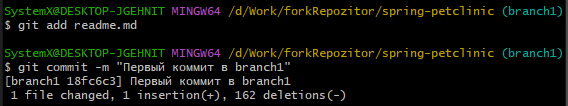


Рисунок 1.5 – Первый коммит для первой ветки

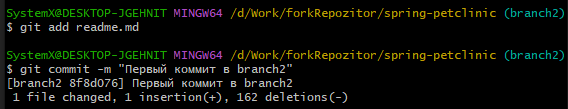


Рисунок 1.6 – Первый коммит для второй ветки

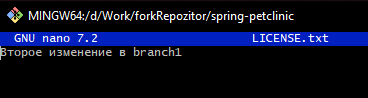


Рисунок 1.7 – Изменение в файле LICENSE.txt

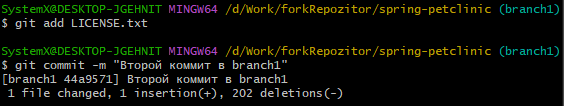


Рисунок 1.8 – Второй коммит для первой ветки

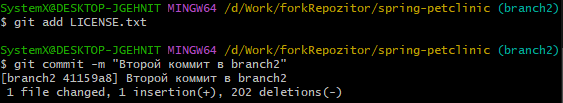


Рисунок 1.9 – Второй коммит для второй ветки

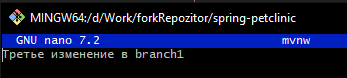


Рисунок 1.10 – Изменение в файле mvnw

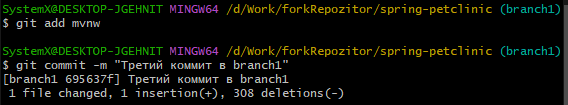


Рисунок 1.11 – Третий коммит для первой ветки

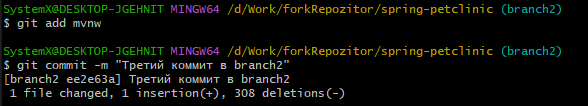


Рисунок 1.12 – Третий коммит для второй ветки

1. Выполним слияние ветки branch1 в ветку branch2.

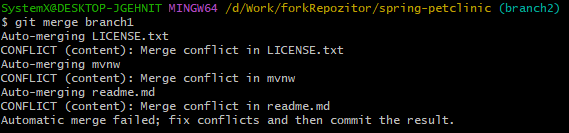
****

Рисунок 1.13 – Ошибки, возникшие при слиянии

При слиянии возникли ошибки. Чтобы исправить их, зайдём в файл, в котором возникла ошибка.

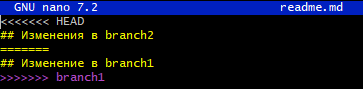


Рисунок 1.14 – Файл с метками, оставленными git

Отредактируем файлы и уберём метки, оставленные git.



Рисунок 1.15 – Отредактированное место в файле readme



Рисунок 1.16 – Отредактированное место в файле mvnw

Сохраним данный файл и введём необходимые команды.



Рисунок 1.17 – Необходимые для разрешения конфликта команды\



Рисунок 1.18 – Успешное слияние после исправления ошибок

1. Выгрузим все изменения во всех ветках в удалённый репозиторий.

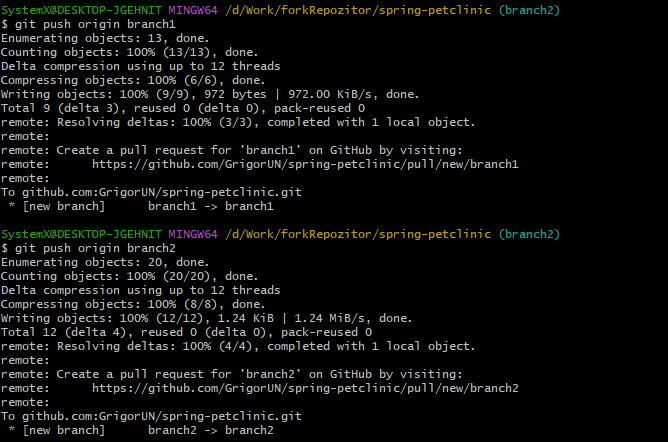


Рисунок 1.19 – Выгрузка изменений в удалённый репозиторий

1. Проведём ещё три коммита в ветку branch1.



Рисунок 1.20 – Изменение в файле readme.md

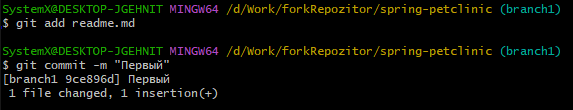


Рисунок 1.21 – Первый новый коммит



Рисунок 1.22 – Изменение в файле LICENSE.txt

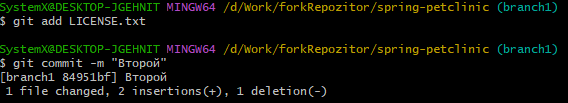


Рисунок 1.22 – Второй новый коммит



Рисунок 1.23 – Изменение в файле mvnw

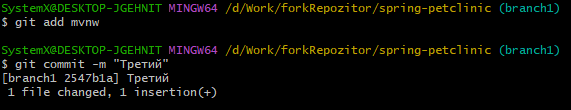


Рисунок 1.24 – Третий новый коммит

1. Склонируем репозиторий ещё раз в другую директорию.

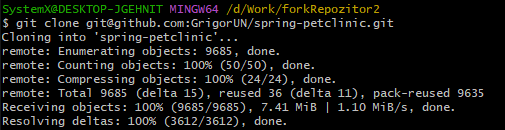


Рисунок 1.25 – Копирование репозиторния в другую директорию

1. В новом клоне рпозитория сделаем три коммита в ветке branch1.



Рисунок 1.26 – Изменение в файле README.md

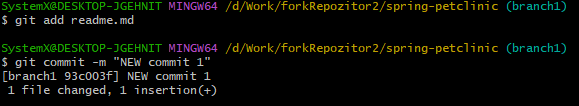


Рисунок 1.27 – Первый коммит в новой копии репозитория

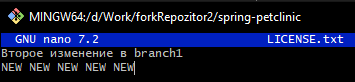


Рисунок 1.28 – Изменение в файле LICENSE.txt

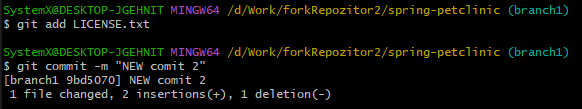


Рисунок 1.29 – Второй коммит в новой копии репозитория



Рисунок 1.30 – Изменение в файле mvnw

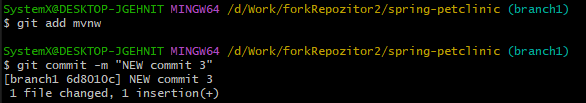


Рисунок 1.31 – Третий коммит в новой копии репозитория

1. Выгрузим все изменения из нового репозитория в удалённый репозиторий.

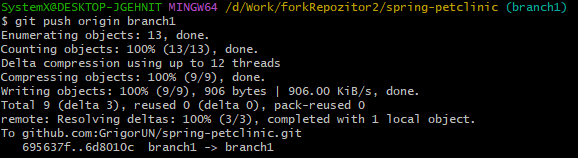


Рисунок 1.32 – Выгрузка изменений из нового репозитория в удалённый

1. Вернёмся в старый клон репозитория и выгрузим изменения с опцией –force.

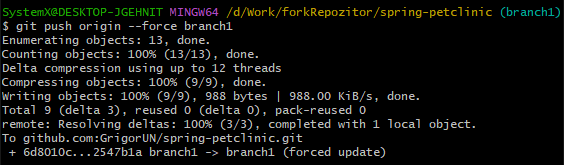


Рисунок 1.33 – Выгрузка изменений с опцией –force

1. Получим все изменения в новом репозитории.

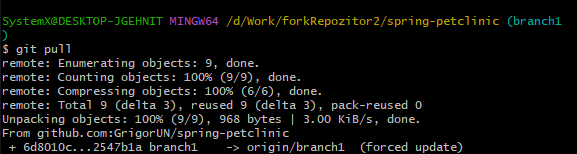


Рисунок 1.34 – Получение изменений в новом репозитории

**Ответы на контрольные вопросы**

1. К какому типу систем контроля версий относится Git?

Git - распределенная система контроля версий, разработанная Линусом Торвальдсом. Изначально Git предназначалась для использования в процессе разработки ядра Linux, но позже стала использоваться и во многих других проектах — таких, как, например, X.org и Ruby on Rails, Drupal. На данный момент Git является самой быстрой распределенной системой, использующей самое компактное хранилище ревизий. Но в тоже время для пользователей, переходящих, например, с Subversion интерфейс Git может показаться сложным.

1. Что такое репозиторий в Git?

Репозиторий - папка проекта, отслеживаемого Git, содержащая дерево изменений проекта в хронологическом порядке. Все файлы истории хранятся в специальной папке .git/ внутри папки проекта.

1. Что такое ветка в репозитории Git?

Ветвь — направление разработки, независимое от других. Ветвь представляет собой копию части (как правило, одного каталога) хранилища, в которую можно вносить свои изменения, не влияющие на другие ветви. Документы в разных ветвях имеют одинаковую историю до точки ветвления и разные — после неё.

1. Что такое тег в репозитории Git?

Метка, которую можно присвоить определённой версии документа. Метка представляет собой символическое имя для группы документов, причём метка описывает не только набор имён файлов, но и версию каждого файла. Версии включённых в метку документов могут принадлежать разным моментам времени.

1. Что такое слияние двух веток?

Слияние — объединение независимых изменений в единую версию документа. Осуществляется, когда два человека изменили один и тот же файл или при переносе изменений из одной ветки в другую.

1. Что делает команда git status?

Команда git status отображает состояние рабочего каталога и раздела проиндексированных файлов. С ее помощью можно проверить индексацию изменений и увидеть файлы, которые не отслеживаются Git.

1. Что означает статус файла modified в выводе команды git status?

Статус изменён (modified) — в рабочей директории есть более новая версия по сравнению с хранящейся в HEAD или в области подготовленных файлов.

1. Что такое Git Hub?

GitHub — веб-сервис, который основан на системе Git. Это такая социальная сеть для разработчиков, которая помогает удобно вести коллективную разработку IT-проектов. Здесь можно публиковать и редактировать свой код, комментировать чужие наработки, следить за новостями других пользователей.

**Вывод**

По итогам выполнения данной практической работы были получены навыки по работе с командной строкой и git’ом.