



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 группы ИКБО-15-22

Оганнисян Г.А.

**Проверил:**

Исобекова О.А.

Москва 2023

## СОДЕРЖАНИЕ

Цель практической работы .....	3
Часть 1. Как начать работу с git'ом? .....	3
1. Установка git на локальную машину.....	3
2. Настройка Git .....	4
3. Начало работы – создание папок и файлов.....	4
4. Создание репозитория.....	5
5. Добавление файла в репозиторий и добавление первого коммита .....	5
6. Индексация изменений.....	6
7. Коммиты нескольких изменений .....	6
8. Просмотр истории коммитов.....	8
9. Получение старых версий.....	8
10. Отмена локальных изменений (до индексации) .....	9
11. Отмена локальных изменений (после индексации и до коммита).....	11
12. Отмена коммита.....	12
Часть 2. Управление репозиториями .....	13
1. Создание SSH-ключа для авторизации.....	13
2. Создание нового репозитория для своего проекта.....	15
3. Связываем локальный и удалённый репозитории.....	15
4. Создание веток и переключение между ними .....	17
5. Слияние веток .....	19
6. Выполнение индивидуального задания.....	19
Часть 3. Выполнение индивидуального задания .....	24
1) Сделаем форк репозитория.....	24
2) Склонируем данный репозиторий на локальную машину .....	24
3) Создадим две ветки branch1 и branch2. ....	24
Проведём по три коммита в каждую из веток, которые меняют один и тот же кусочек файла.....	25
4) Выполним слияние ветки branch1 в ветку branch2.....	26
5) Выгрузим все изменения во всех ветках в удалённый репозиторий.....	27
6) Проведём ещё три коммита в ветку branch1 .....	28
7) Склонируем репозиторий ещё раз в другую директорию .....	28
8) В новом клоне репозитория сделаем три коммита в ветке branch1.....	29
9) Выгрузим все изменения из нового репозитория в удалённый репозиторий.....	30
10) Вернёмся в старый клон репозитория и выгрузим изменения с опцией –force.....	30
11) Получим все изменения в новом репозитории .....	30
Ответы на контрольные вопросы.....	31
Вывод.....	33

## 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>).

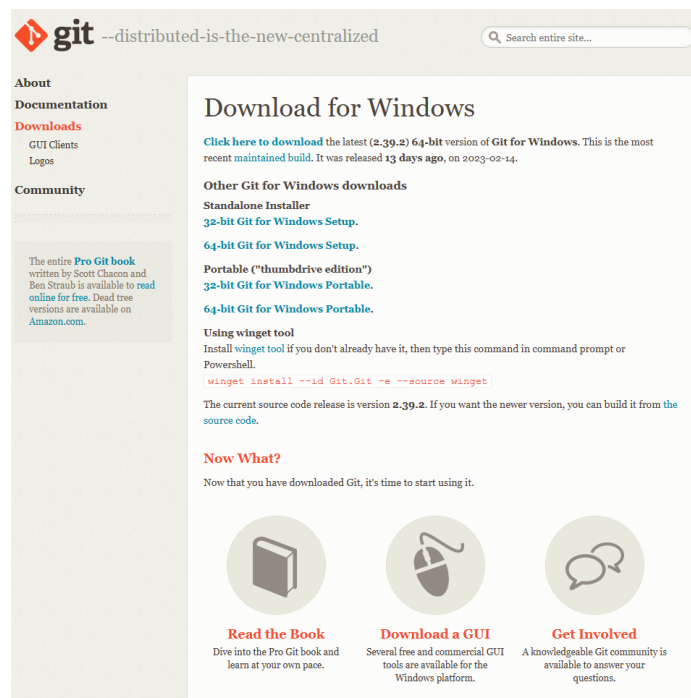


Рисунок 1.1 – Сайт git

Теперь запустим установщик.

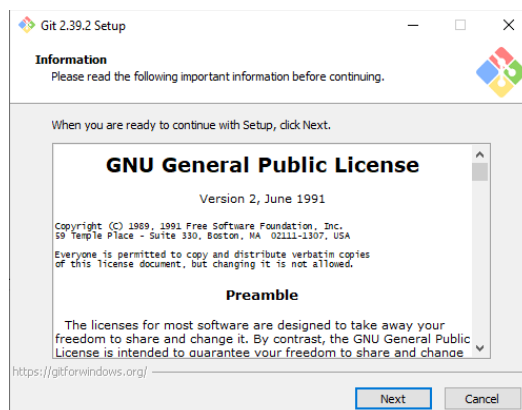
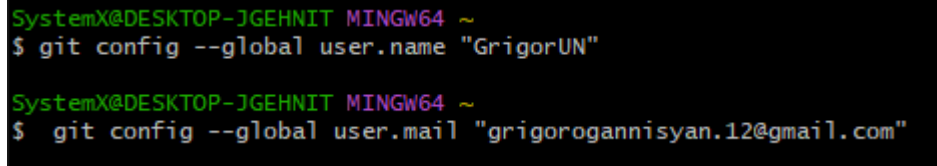


Рисунок 1.2 – Запущенный установщик git

## 2. Настройка Git

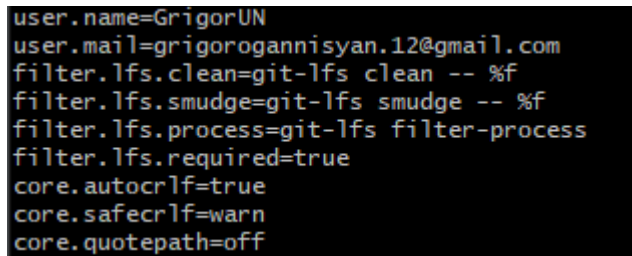
Откроем терминал и выполним две команды.



```
SystemX@DESKTOP-JGEHNIT MINGW64 ~  
$ git config --global user.name "GrigorUN"  
  
SystemX@DESKTOP-JGEHNIT MINGW64 ~  
$ git config --global user.mail "grigorogannisyan.12@gmail.com"
```

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 введенных данных при помощи команды `git config --list`.

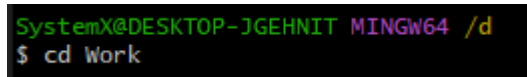


```
user.name=GrigorUN  
user.mail=grigorogannisyan.12@gmail.com  
filter.lfs.clean=git-lfs clean -- %f  
filter.lfs.smudge=git-lfs smudge -- %f  
filter.lfs.process=git-lfs filter-process  
filter.lfs.required=true  
core.autocrlf=true  
core.safecrlf=warn  
core.quotePath=off
```

Рисунок 2.2 – Проверка правильности введенных данных

## 3. Начало работы – создание папок и файлов

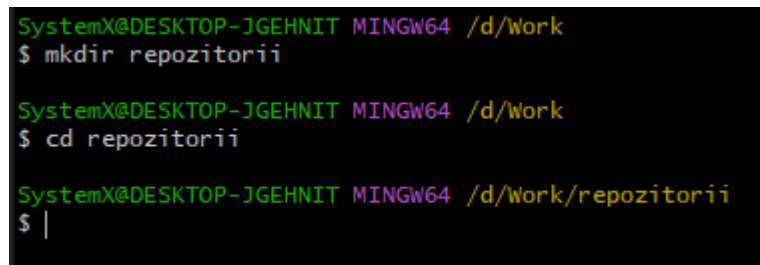
Для перехода в другую директорию воспользуемся командой `cd <путь к директории>`. Перейдём в директорию `Documents`.



```
SystemX@DESKTOP-JGEHNIT MINGW64 /d  
$ cd Work
```

Рисунок 3.1 – Переход в директорию Work

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



```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work  
$ mkdir repozitorii  
  
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work  
$ cd repozitorii  
  
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repozitorii  
$ |
```

Рисунок 3.2 – Создание папки repozitorii

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

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repositorii
$ touch proekt.html
```

Рисунок 3.3 – Создание файла `proekt.html`

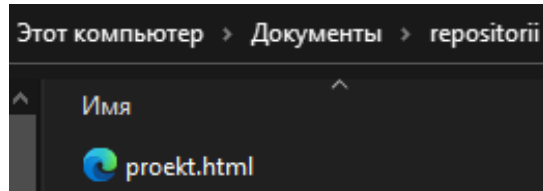


Рисунок 3.4 – Проверка наличия созданного файла

Воспользуемся командой `nano` для добавления содержания в файл.

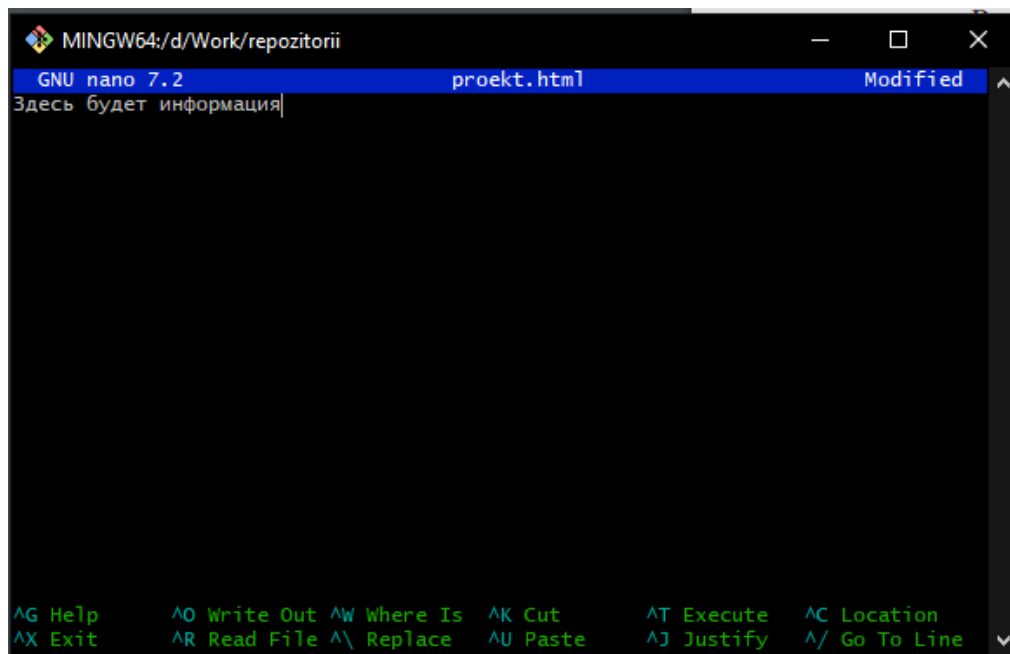


Рисунок 3.5 – Переход в файл командой `nano`

#### 4. Создание репозитория

Для создания репозитория выполним команду `git init`.

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repositorii
$ git init
Initialized empty Git repository in D:/Work/repositorii/.git/
```

Рисунок 4.1 – Создание нового репозитория

#### 5. 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"`.

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repositorii (master)
$ git commit -m "Наш первый коммит"
[master (root-commit) 3611982] Наш первый коммит
1 file changed, 1 insertion(+)
create mode 100644 proekt.html
```

Рисунок 5.1 – Добавление коммита

Выполним команду git status.

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repositorii (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Рисунок 5.2 – Выполнение команды git status

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

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repositorii (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   proekt.html

no changes added to commit (use "git add" and/or "git commit -a")
```

Рисунок 5.3 – Проверка внесения изменений в файл

## 6. Индексация изменений

Выполним команды git add proekt.html и git status.

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repositorii (master)
$ git add proekt.html

SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repositorii (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   proekt.html
```

Рисунок 6.1 – Индексация изменений

## 7. Коммиты нескольких изменений

При помощи команды nano внесём изменения в файл.

```
MINGW64:/d/Work/repositorii
GNU nano 7.2 proekt
<html>
  <body>
    <h1>Hello, World!</h1>
  </body>
</html>
```

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`.

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repozitorii (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   projekt.html

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   projekt.html
```

Рисунок 7.2 – Результат выполнения команды `git status`

Добавим коммит и проверим статус репозитория.

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repozitorii (master)
$ git commit -m "Added standard HTML page tags"
[master b7f2c0b] Added standard HTML page tags
1 file changed, 5 insertions(+), 1 deletion(-)

SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repozitorii (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   projekt.html

no changes added to commit (use "git add" and/or "git commit -a")
```

Рисунок 7.3 – Добавление коммита и проверка статуса репозитория

Выполним команды `git add .` и `git status`.

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repozitorii (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   projekt.html
```

Рисунок 7.4 – Результат выполнения двух команд

Далее делаем коммит второго изменения.

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repozitorii (master)
$ git commit -m "Added HTML header"
[master 101a986] Added HTML header
1 file changed, 2 insertions(+)
```

Рисунок 7.5 – Коммит второго изменения

## 8. Просмотр истории коммитов

Для просмотра истории коммитов используется команда `git log`.

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repositorii (master)
$ git log
commit 101a98680587e7b1d19631f4e200e336490eb260 (HEAD -> master)
Author: GrigorUN <garikogannisyan.9@gmail.com>
Date: Mon Feb 12 14:06:50 2024 +0300

    Added HTML header

commit b7f2c0b6ccb40763609ba8066502fc50b069d9fd
Author: GrigorUN <garikogannisyan.9@gmail.com>
Date: Mon Feb 12 14:04:17 2024 +0300

    Added standard HTML page tags

commit 361198209b8c9bca11b5be4c2ece62f811f756bf
Author: GrigorUN <garikogannisyan.9@gmail.com>
Date: Mon Feb 12 13:52:56 2024 +0300

    Наш первый коммит
```

Рисунок 8.1 – Просмотр истории коммитов

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

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repositorii (master)
$ git log --pretty=oneline
101a98680587e7b1d19631f4e200e336490eb260 (HEAD -> master) Added HTML header
b7f2c0b6ccb40763609ba8066502fc50b069d9fd Added standard HTML page tags
361198209b8c9bca11b5be4c2ece62f811f756bf Наш первый коммит
```

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`.

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repositorii (master)
$ git log --pretty=format:"%h %ad | %s%d [%an]" --graph --date=short
* 101a986 2024-02-12 | Added HTML header (HEAD -> master) [GrigorUN]
* b7f2c0b 2024-02-12 | Added standard HTML page tags [GrigorUN]
* 3611982 2024-02-12 | Наш первый коммит [GrigorUN]
```

Рисунок 8.3 – Результат выполнения команды

## 9. Получение старых версий

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

Теперь введём команду `git checkout <хэш>`.



```

SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repositorii (master)
$ git checkout 3611982
Note: switching to '3611982'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

    git switch -c <new-branch-name>

Or undo this operation with:

    git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 3611982 Наш первый коммит

```

Рисунок 9.1 – Результат выполнения команды git checkout

Проверим наличие данных в файле командой cat proekt.html

```

SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repositorii ((3611982...))
$ cat proekt.html
Здесь будет информация

```

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.

```

SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repositorii ((3611982...))
$ git checkout master
Previous HEAD position was 3611982 Наш первый коммит
Switched to branch 'master'

SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repositorii (master)
$ cat proekt.html
<html>
  <head>
  </head>
  <body>
    <h1>Hello, World!</h1>
  </body>
</html>

```

Рисунок 9.3 – Возврат в текущее состояние файла

## 10. 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.

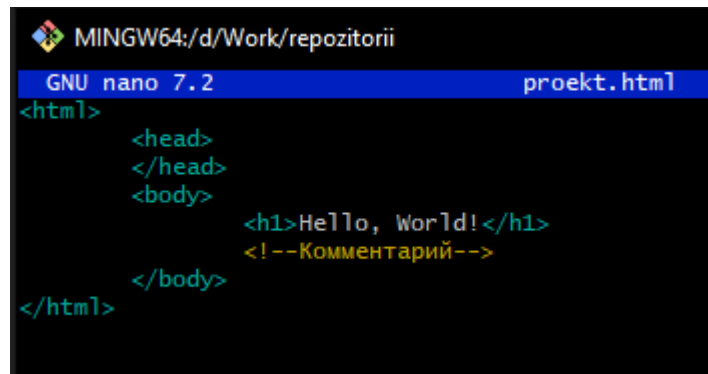
```

SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repositorii (master)
$ git checkout master
Already on 'master'

```

Рисунок 10.1 – Проверка нахождения на последнем коммите ветки master

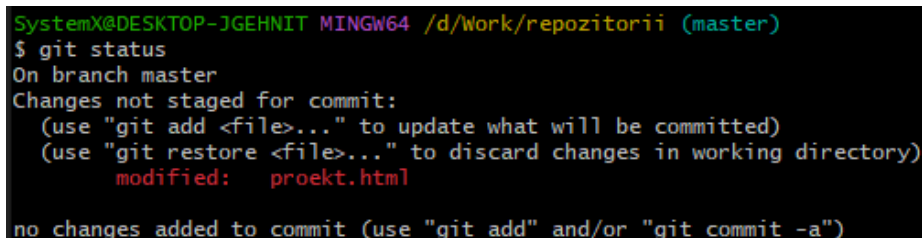
Изменим наш файл в рабочем каталоге при помощи команды nano и сохраним его.



```
MINGW64:/d/Work/repozitorii
GNU nano 7.2 projekt.html
<html>
  <head>
  </head>
  <body>
    <h1>Hello, World!</h1>
    <!--Комментарий-->
  </body>
</html>
```

Рисунок 10.2 – Внесение изменений в файл

Проверим состояние рабочего каталога при помощи команды git status.

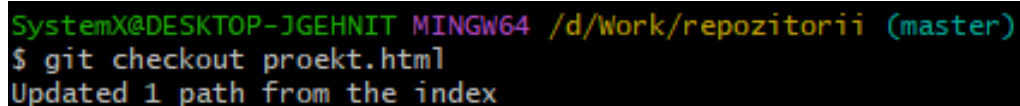


```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repozitorii (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   projekt.html

no changes added to commit (use "git add" and/or "git commit -a")
```

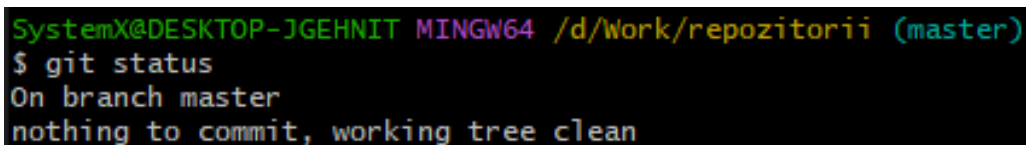
Рисунок 10.3 – Проверка состояния рабочего каталога

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



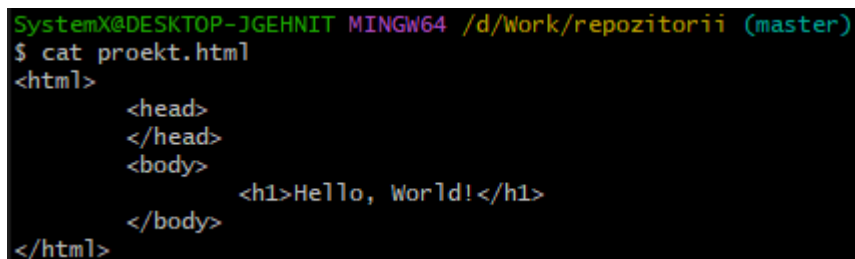
```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repozitorii (master)
$ git checkout projekt.html
Updated 1 path from the index
```

Рисунок 10.4 - Результат выполнения команды git checkout



```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repozitorii (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Рисунок 10.5 – Результат выполнения команды git status

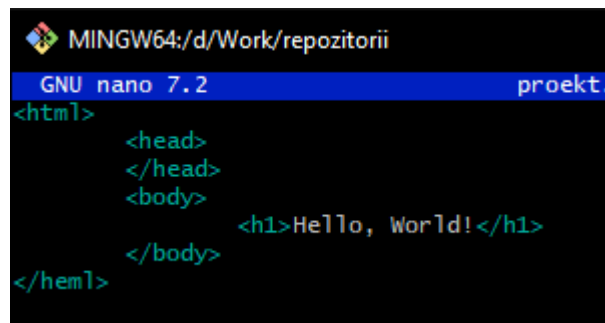


```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repozitorii (master)
$ cat projekt.html
<html>
  <head>
  </head>
  <body>
    <h1>Hello, World!</h1>
  </body>
</html>
```

Рисунок 10.6 – Проверка отмены локальных изменений

## 11. Отмена локальных изменений (после индексации и до коммита)

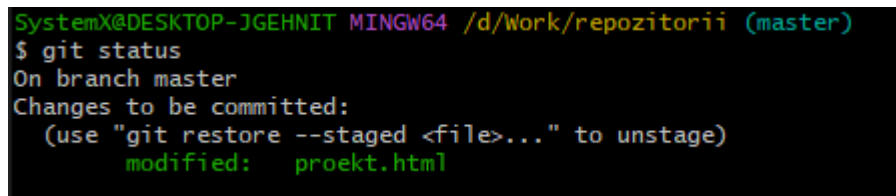
Снова внесём изменения в файл.



```
MINGW64:/d/Work/repozitorii
GNU nano 7.2                                proekt.html
<html>
    <head>
    </head>
    <body>
        <h1>Hello, World!</h1>
    </body>
</html>
```

Рисунок 11.1 – Внесение изменений в файл

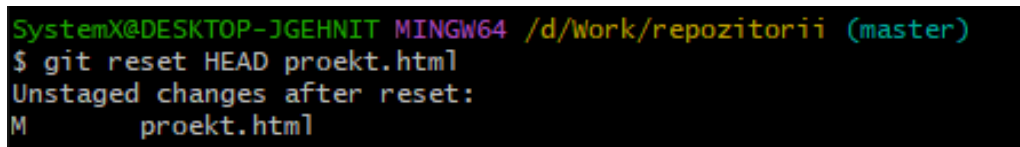
Проиндексируем наши изменения с помощью команды `git add` и проверим состояние нашего изменения при помощи команды `git status`.



```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repozitorii (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   proekt.html
```

Рисунок 11.2 – Проверка состояния нашего изменения

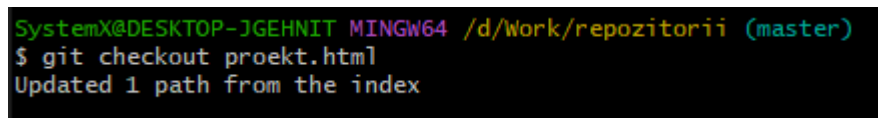
Отменим индексацию изменений.



```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repozitorii (master)
$ git reset HEAD proekt.html
Unstaged changes after reset:
M       proekt.html
```

Рисунок 11.3 – Отмена индексации изменений

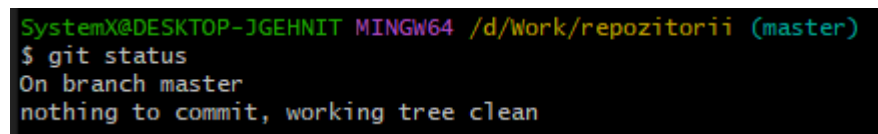
Уберём ненужный комментарий при помощи команды `git checkout`.



```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repozitorii (master)
$ git checkout proekt.html
Updated 1 path from the index
```

Рисунок 11.4 – Удаление ненужного комментария

Выполним команду `git status` для проверки статуса рабочего каталога.

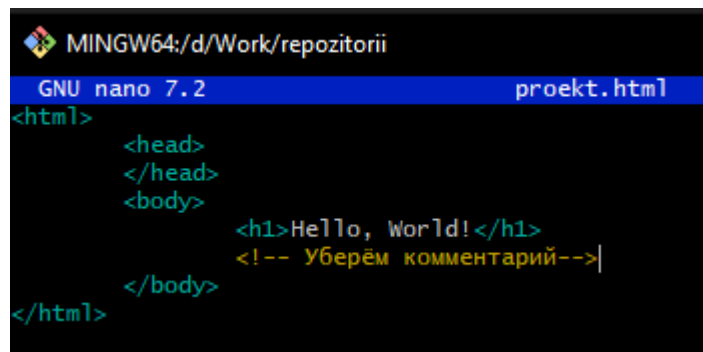


```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repozitorii (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Рисунок 11.5 – Результат проверки статуса рабочего каталога

## 12.Отмена коммита

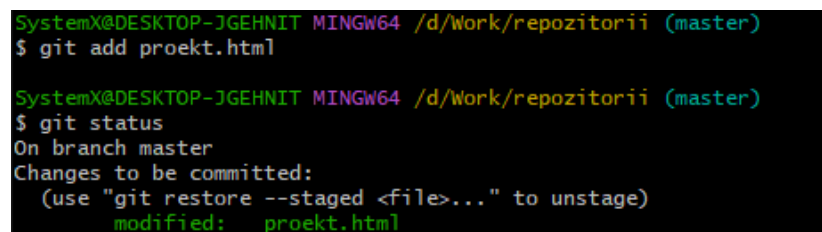
Внесем снова изменения в наш файл при помощи команды nano.



```
MINGW64:/d/Work/repozitorii
GNU nano 7.2 proekt.html
<html>
  <head>
  </head>
  <body>
    <h1>Hello, World!</h1>
    <!-- Уберём комментарий-->|
  </body>
</html>
```

Рисунок 12.1 – Внесение изменений в файл

Проиндексируем наше изменение при помощи команды git add.

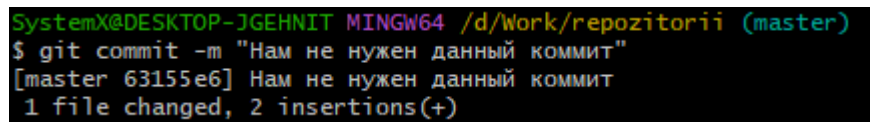


```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repozitorii (master)
$ git add proekt.html

SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repozitorii (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   proekt.html
```

Рисунок 12.2 – Индексация изменения

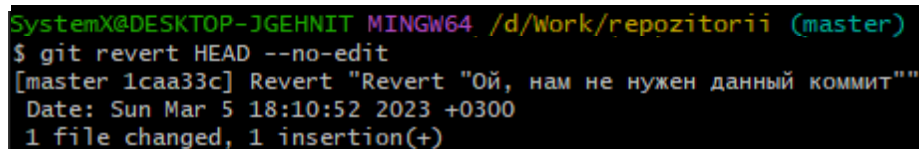
Произведем следующий коммит при помощи команды git commit -m.



```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repozitorii (master)
$ git commit -m "Нам не нужен данный коммит"
[master 63155e6] Нам не нужен данный коммит
1 file changed, 2 insertions(+)
```

Рисунок 12.3 – Добавление коммита

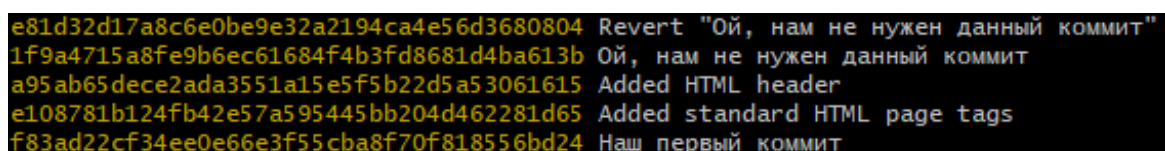
Чтобы отменить коммит, нам необходимо сделать коммит, который удаляет изменения, сохраненные нежелательным коммитом, для это воспользуемся следующей командой git revert HEAD --no-edit.



```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repozitorii (master)
$ git revert HEAD --no-edit
[master 1caa33c] Revert "Revert "Ой, нам не нужен данный коммит""
Date: Sun Mar 5 18:10:52 2023 +0300
1 file changed, 1 insertion(+)
```

Рисунок 12.4 – Отмена коммита

Проверим историю коммитов.



```
e81d32d17a8c6e0be9e32a2194ca4e56d3680804 Revert "Ой, нам не нужен данный коммит"
1f9a4715a8fe9b6ec61684f4b3fd8681d4ba613b Ой, нам не нужен данный коммит
a95ab65dece2ada3551a15e5f5b22d5a53061615 Added HTML header
e108781b124fb42e57a595445bb204d462281d65 Added standard HTML page tags
f83ad22cf34ee0e66e3f55cba8f70f818556bd24 Наш первый коммит
```

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.

```
SystemX@DESKTOP-JGEHNIT MINGW64 ~  
$ cd ~/.ssh  
  
SystemX@DESKTOP-JGEHNIT MINGW64 ~/.ssh
```

### Рисунок 1.1 – Переход в каталог с ssh-ключами

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

```
SystemX@DESKTOP-JGEHNIT MINGW64 ~  
$ ssh-keygen -t rsa -b 4096 -C "grigorogannisyan.12@gmail.com"  
Generating public/private rsa key pair.  
Enter file in which to save the key (/c/Users/SystemX/.ssh/id_rsa):  
Created directory '/c/Users/SystemX/.ssh'.  
Enter passphrase (empty for no passphrase):  
Enter same passphrase again:  
Your identification has been saved in /c/Users/SystemX/.ssh/id_rsa  
Your public key has been saved in /c/Users/SystemX/.ssh/id_rsa.pub  
The key fingerprint is:  
SHA256:/D5GT3fnPq+e4mnfNhWW64rvDx9rLVRPkv6tc/QMsQA grigorogannisyan.12@gmail.com  
The key's randomart image is:  
+----[RSA 4096]-----+  
|          E          |  
|      .    ..       |  
|.   .  . ++o        |  
| S     o.*+         |  
|   . . *. *         |  
|   .o +.O*          |  
|   .o =.=B%         |  
|   ..==BX%B         |  
+----[SHA256]-----+
```

### Рисунок 1.2 – Генерация и сохранение ssh-ключа

Добавляем ключ в shh-agent.

```
SystemX@DESKTOP-JGEHNIT MINGW64 ~/.ssh
$ ssh-agent
SSH_AUTH_SOCK=/tmp/ssh-pmmUnh0XPSeB/agent.1853; export SSH_AUTH_SOCK;
SSH_AGENT_PID=1854; export SSH_AGENT_PID;
echo Agent pid 1854;
```

### Рисунок 1.3 – Добавление ключа в ssh-agent

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

```
SystemX@DESKTOP-JGEHNIT MINGW64 ~/.ssh
$ eval "$(ssh-agent -s)"
Agent pid 1859
```

### 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.

```
SystemX@DESKTOP-JGEHNIT MINGW64 ~/.ssh
$ ssh-add ~/.ssh/id_rsa
Identity added: /c/Users/SystemX/.ssh/id_rsa (grigorogannisyan.12@gmail.com)
```

Рисунок 1.5 – Добавление имени ключа

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

```
SystemX@DESKTOP-JGEHNIT MINGW64 ~/.ssh
$ cat id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQADexM1xfpNFuBnVdyCY8Wv1Df6QNm2CZd9g39Dju+HKmN9ZwkDo3JuYemkXqd6AbTLEbf8lhyZR54bu2sFxDa7/ALvnZ9i+Ked2oURmTGEgw5WwmlHQBI568Ui+h6lKW8h2pLIVxfg4TA/9GH6PqJ8J9iDwKX6q4Ge5f2zpotWbd1ErdZ03LnhgvYZ9xmzWrf9i0SZIkVmawpX3k9VwdKYOCe4A+gbHjz1+xT93tVsSjw7rwiQzevoWwXPR37INPcLnn5c1fHbmPNxwhUJg9W3OcVgDKN5BITDimKRV6sW2N46qB115/W0xOs0d3Z1eIp8htz4jwp+JBSP8lBvyzcapJbjBwobHCXMsThJQyn195/H1seF5+b/98t0MAgWqCLk0AjM0i5yx2rY07uUrvHf+3C0c330A55swjNPPD44tvDgb5o2QjSh2YnAjmL5tEPcKi5VA5rfvx7wAeOuGBDDI5/1wLdpryu5d5cwlCoq4MOyU/INhciPeaWxcAXEjBgOuxxu52Zxjno2Q4Obqlc7dX01IuyMJKjTSiHpa1TEPswia7VaCMcaom38Tw4brJlI2BYitrwuQ6z9le2Fhf9j427Z31Ws2IX6x+X+KNDqR6ea7PXanYFLitUoqq4agxywuBr4XK/7WwBSCHZbw2Mvcq5OrW/F0yvLiTOJ2w== grigorogannisyan.12@gmail.com
```

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”.

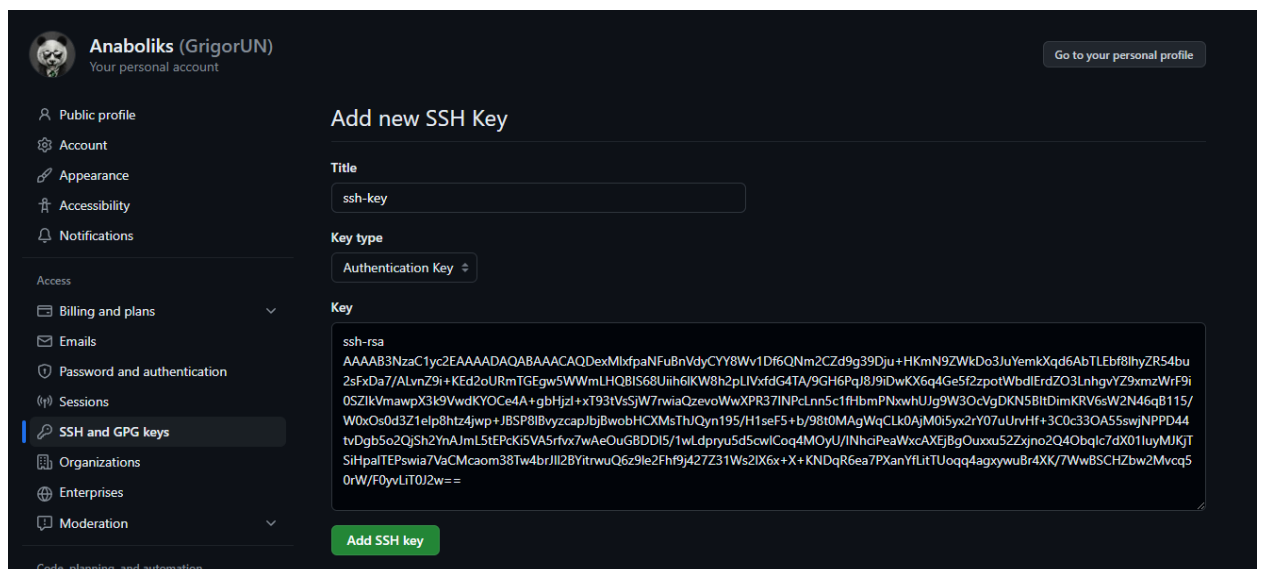


Рисунок 1.7 – Добавление ключа в Git Hub

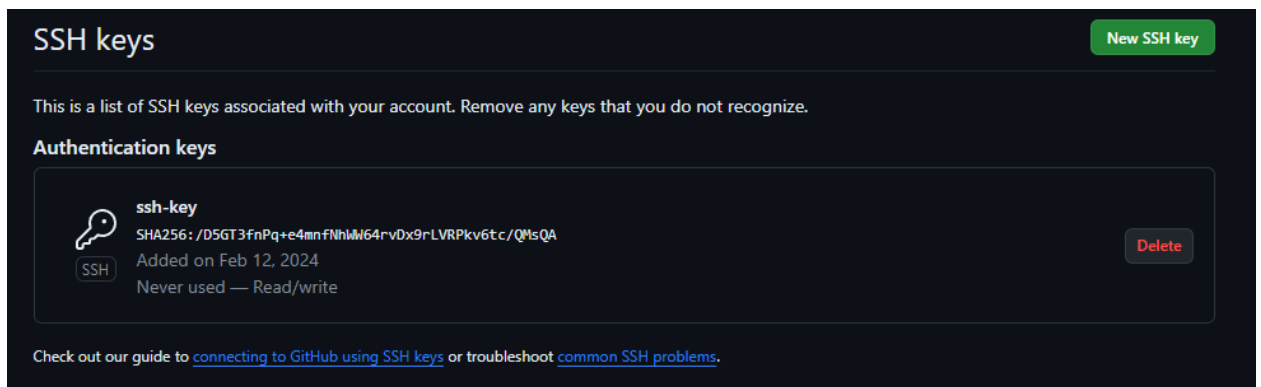


Рисунок 1.8 – Добавленный ключ

## 2. Создание нового репозитория для своего проекта

Создадим новую папку проекта в другой директории.

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work
$ mkdir repozitor

SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work
$ cd repozitor/

SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repozitor
```

Рисунок 2.1 – Создание новой папки проекта

Теперь добавим несколько файлов.

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repozitor
$ touch readme.txt

SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repozitor
$ touch main.py

SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repozitor
$ touch fn.py
```

Рисунок 2.2 – Добавление файлов в папку проекта

После добавления файлов создадим новый локальный репозиторий.

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repozitor
$ git init
Initialized empty Git repository in D:/Work/repozitor/.git/

SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repozitor (master)
```

Рисунок 2.3 – Создание нового локального репозитория

## 3. Связываем локальный и удалённый репозитории

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

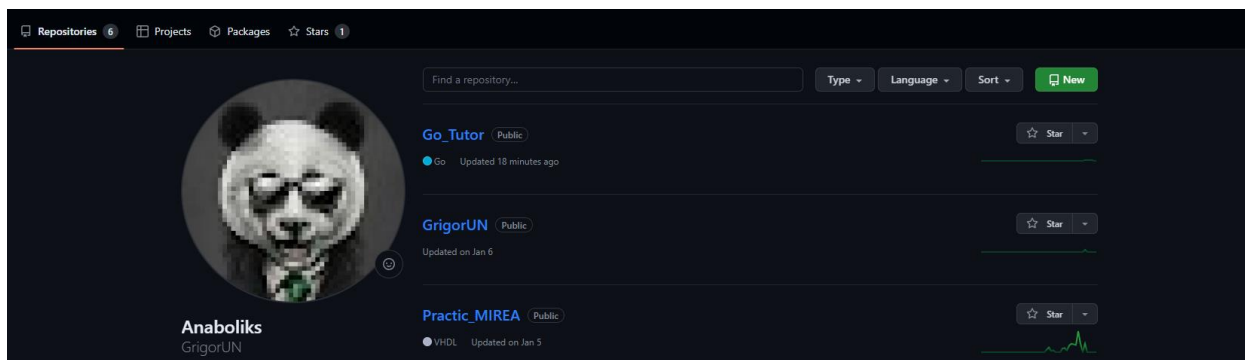


Рисунок 3.1 – Вкладка с репозиториями

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

### 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 \*** GrigorUN / **Repository name \*** TRPP\_PR1  
 ✓ TRPP\_PR1 is available.

Great repository names are short and memorable. Need inspiration? How about **probable-goggles** ?

**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: **None**

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 3.2 – Окно создания репозитория



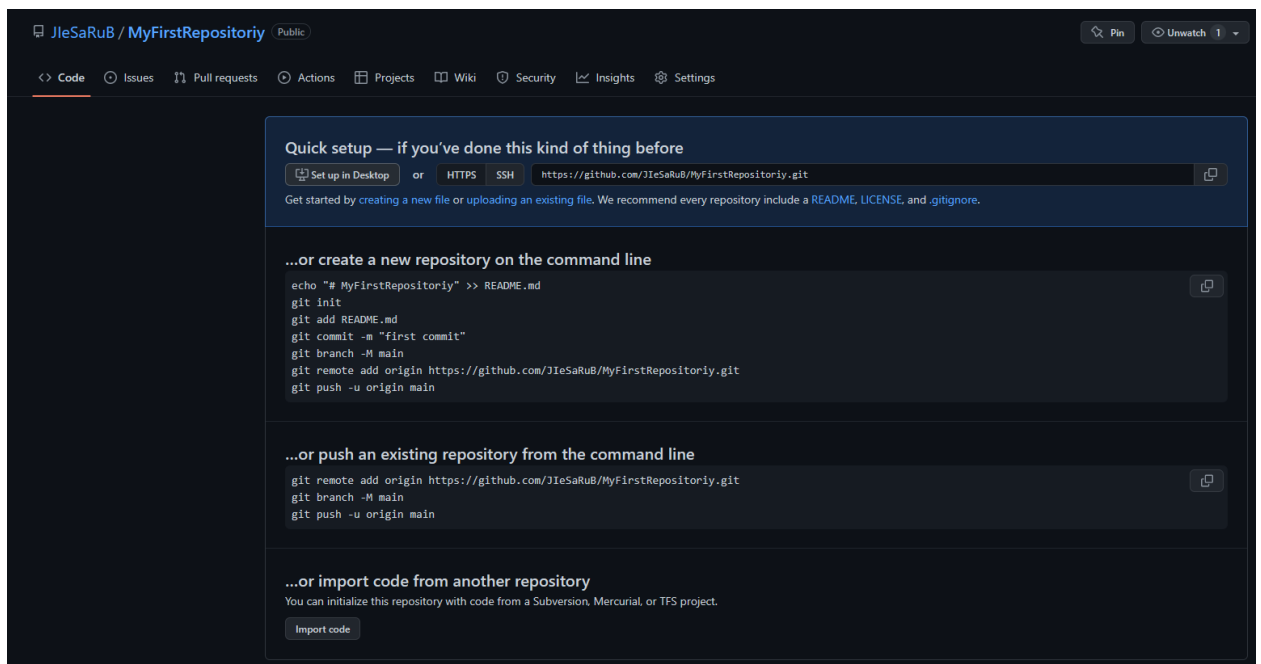


Рисунок 3.3 – Созданный пустой репозиторий

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

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repozitor (master)
$ git remote add project git@github.com:GrigorUN/TRPP_PR1
```

Рисунок 3.4 – Связывание локального и удалённого репозитория

#### 4. Создание веток и переключение между ними

Для создания новой ветки необходимо воспользоваться командой `git checkout -b`.

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repozitor (master)
$ git checkout -b main
Switched to a new branch 'main'

SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repozitor (main)
$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    fn.py
    main.py
    readme.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Рисунок 4.1 – Создание новой ветки

Теперь внесём некоторые изменения в файлы.

```
MINGW64:/d/Work/repozitor
GNU nano 7.2 main.py
s = [1,2,3,4]
print(s)
```

Рисунок 4.2 – Изменения в файле main.py

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repozitor (main)
$ git commit -m "Вывод s"
[main (root-commit) 576eb73] Вывод s
1 file changed, 2 insertions(+)
create mode 100644 main.py
```

Рисунок 4.3 – Коммит для файла main.py

```
MINGW64:/d/Work/repozitor
GNU nano 7.2 fn.py
for i in range(1, 16, 2):
    print(i)
```

Рисунок 4.4 – Изменения в файле fn.py

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repozitor (main)
$ git commit -m "Вывод нечётных от 1 до 16"
[main 83330db] Вывод нечётных от 1 до 16
1 file changed, 2 insertions(+)
create mode 100644 fn.py
```

Рисунок 4.5 – Коммит для файла fn.py

```
MINGW64:/d/Work/repozitor
GNU nano 7.2 readme.txt
Два маленьких кода на Python
```

Рисунок 4.6 – Изменение в файле readme.txt

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repozitor (main)
$ git commit -m "Добавление информации в readme"
[main ed2ed51] Добавление информации в readme
1 file changed, 1 insertion(+)
create mode 100644 readme.txt
```

Рисунок 4.7 – Коммит для файла readme.txt

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repozitor (main)
$ git log --pretty=oneline
ed2ed51192ca470b9204b3f2f92e623720f839ec (HEAD -> main) Добавление информации в
readme
83330dbe84a7501797dd3d1fd0a6ad4a944f5379 Вывод нечётных от 1 до 16
576eb73319a225cb448c6a9ba9815ed24512713c Вывод s
```

Рисунок 4.8 – Проверка наличия всех коммитов

## 5. Слияние веток

Создадим дополнительную ветку addition.

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repozitor (main)
$ git checkout -b addition
Switched to a new branch 'addition'
```

Рисунок 5.1 – Создание новой ветки

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/repozitor (addition)
$ git merge main
Already up to date.
```

Рисунок 5.2 – Слияние веток

## 6. Выполнение индивидуального задания Вариант №9.

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

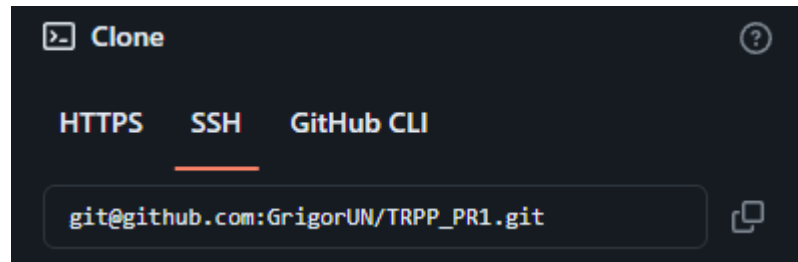


Рисунок 6.1 – Ссылка на удалённый репозиторий

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/cloneRepozitor
$ git clone git@github.com:GrigorUN/TRPP_PR1.git
Cloning into 'TRPP_PR1'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.
```

Рисунок 6.2 – Клонирование удалённого репозитория на локальную машину

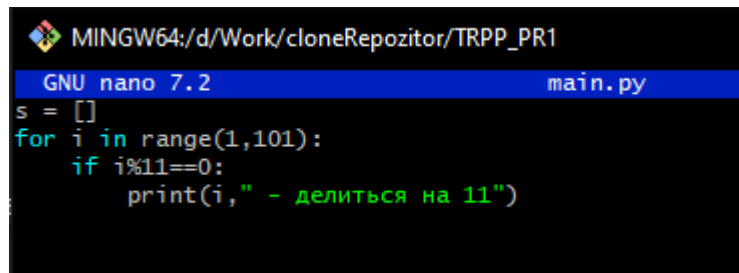
2) Теперь создадим новую ветку и выведем список всех веток.

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/cloneRepozitor/TRPP_PR1 (main)
$ git branch addition

SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/cloneRepozitor/TRPP_PR1 (main)
$ git branch --list
  addition
* main
```

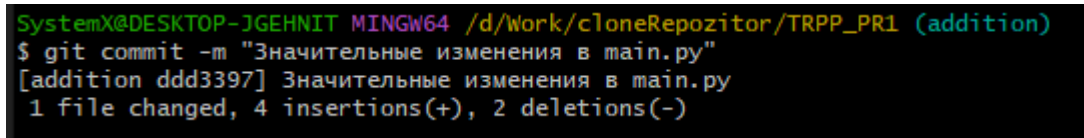
Рисунок 6.3 – Создание новой ветки и вывод списка веток

3) Произведём 5 коммита на разные файлы в новой ветке.



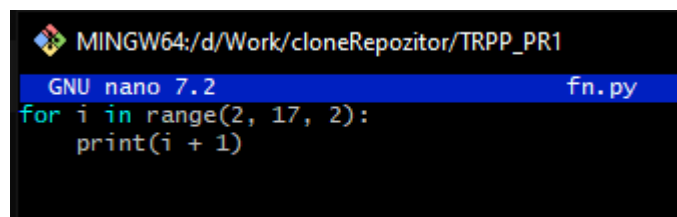
```
MINGW64:/d/Work/cloneRepozitor/TRPP_PR1
GNU nano 7.2 main.py
s = []
for i in range(1,101):
    if i%11==0:
        print(i," - делиться на 11")
```

Рисунок 6.4 – Изменение в файле main.py



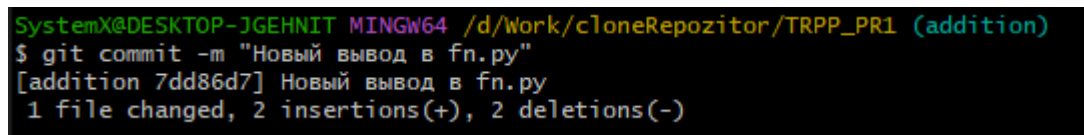
```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/cloneRepozitor/TRPP_PR1 (addition)
$ git commit -m "Значительные изменения в main.py"
[addition ddd3397] Значительные изменения в main.py
1 file changed, 4 insertions(+), 2 deletions(-)
```

Рисунок 6.5 – Добавление первого коммита



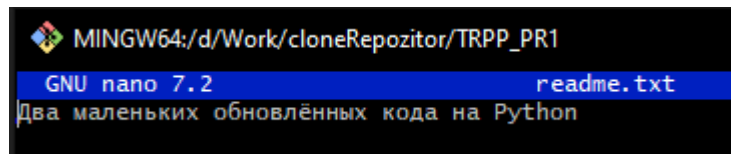
```
MINGW64:/d/Work/cloneRepozitor/TRPP_PR1
GNU nano 7.2 fn.py
for i in range(2, 17, 2):
    print(i + 1)
```

Рисунок 6.6 – Изменение в файле fn.py



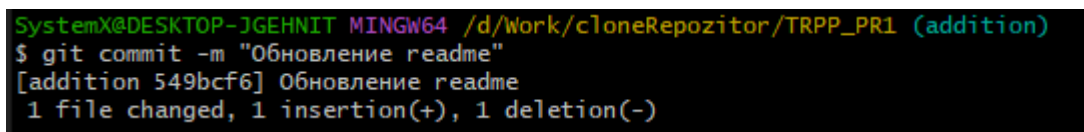
```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/cloneRepozitor/TRPP_PR1 (addition)
$ git commit -m "Новый вывод в fn.py"
[addition 7dd86d7] Новый вывод в fn.py
1 file changed, 2 insertions(+), 2 deletions(-)
```

Рисунок 6.7 – Добавление второго коммита



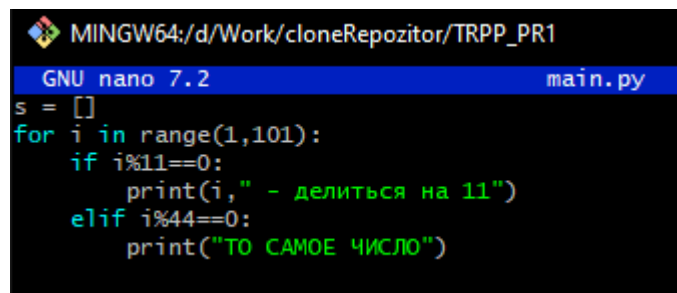
```
MINGW64:/d/Work/cloneRepozitor/TRPP_PR1
GNU nano 7.2 readme.txt
Два маленьких обновлённых кода на Python
```

Рисунок 6.8 – Изменение в файле readme.txt



```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/cloneRepozitor/TRPP_PR1 (addition)
$ git commit -m "Обновление readme"
[addition 549bcf6] Обновление readme
1 file changed, 1 insertion(+), 1 deletion(-)
```

Рисунок 6.9 – Добавление третьего коммита



```
MINGW64:/d/Work/cloneRepozitor/TRPP_PR1
GNU nano 7.2 main.py
s = []
for i in range(1,101):
    if i%11==0:
        print(i," - делиться на 11")
    elif i%44==0:
        print("ТО САМОЕ ЧИСЛО")
```

Рисунок 6.10 – Дополнительное изменение в файле main.py

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/cloneRepozitor/TRPP_PR1 (addition)
$ git commit -m "Доп. изменение в main.py"
[addition 880208f] Доп. изменение в main.py
1 file changed, 3 insertions(+), 1 deletion(-)
```

Рисунок 6.11 – Добавление четвёртого коммита

```
MINGW64:/d/Work/cloneRepozitor/TRPP_PR1
GNU nano 7.2 fn.py
for i in range(2, 45, 2):
    if i == 28:
        break
    print(i + 1)
```

Рисунок 6.12 – Дополнительное изменение в файле fn.py

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/cloneRepozitor/TRPP_PR1 (addition)
$ git commit -m "Улучшение fn.py"
[addition dd51d13] Улучшение fn.py
1 file changed, 3 insertions(+), 1 deletion(-)
```

Рисунок 6.13 – Добавление пятого коммита

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/cloneRepozitor/TRPP_PR1 (addition)
$ git log --pretty=oneline
dd51d133ae2174bb38ce62c935ee820818d987d2 (HEAD -> addition) Улучшение fn.py
880208fb8e1f4e174a3e8cd568690c149111eae4 Доп. изменение в main.py
549bcf6360f036b8ff11d50f786235f89dcc56dd Обновление readme
7dd86d7b93906c2fe778be12d4c9fcb30093f971 Новый вывод в fn.py
ddd3397ed755a2dd41da0747b805aa01059bd153 Значительные изменения в main.py
1679a4a1faeac0b9610ca2e0e4e847b0e5b9ee75 (origin/main, origin/HEAD, main) 1
```

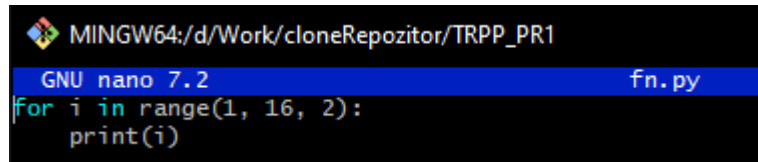
Рисунок 6.14 – Проверка наличия всех коммитов

4) Выгрузим изменения в удалённый репозиторий. Выгрузка в удалённый репозиторий выполняется с помощью команды `git push origin`.

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/cloneRepozitor/TRPP_PR1 (addition)
$ git push origin addition
Enumerating objects: 19, done.
Counting objects: 100% (19/19), done.
Delta compression using up to 12 threads
Compressing objects: 100% (14/14), done.
Writing objects: 100% (15/15), 1.77 KiB | 1.77 MiB/s, done.
Total 15 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), done.
remote:
remote: Create a pull request for 'addition' on GitHub by visiting:
remote:   https://github.com/GrigorUN/TRPP_PR1/pull/new/addition
remote:
To github.com:GrigorUN/TRPP_PR1.git
 * [new branch]      addition -> addition
```

Рисунок 6.15 – Выгрузка изменений в удалённый репозиторий

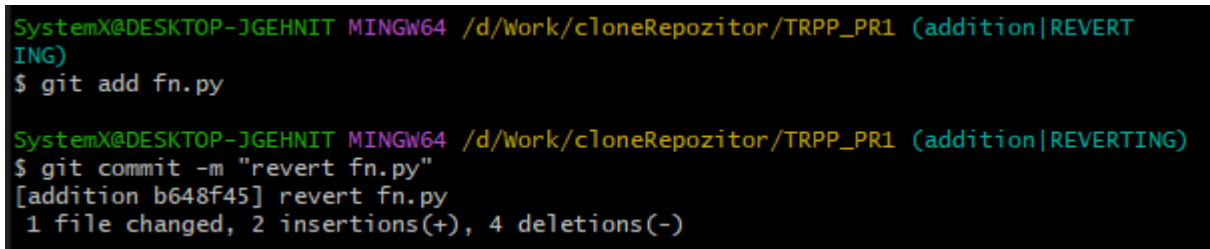
- 5) Произведём revert 2-го и 4-го коммита в новой ветке.



```
MINGW64:/d/Work/cloneRepozitor/TRPP_PR1
GNU nano 7.2 fn.py
for i in range(1, 16, 2):
    print(i)
```

Рисунок 6.16 – Revert 2-го коммита.

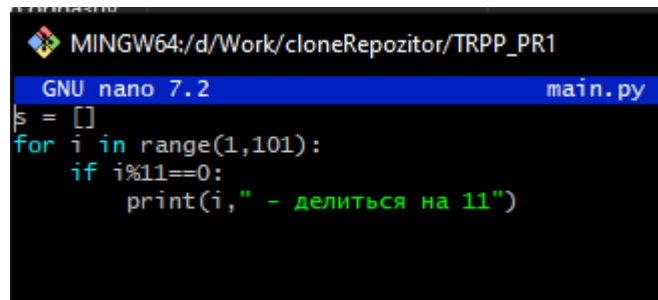
Проиндексируем изменения и добавим новый коммит



```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/cloneRepozitor/TRPP_PR1 (addition|REVERTING)
$ git add fn.py

SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/cloneRepozitor/TRPP_PR1 (addition|REVERTING)
$ git commit -m "revert fn.py"
[addition b648f45] revert fn.py
1 file changed, 2 insertions(+), 4 deletions(-)
```

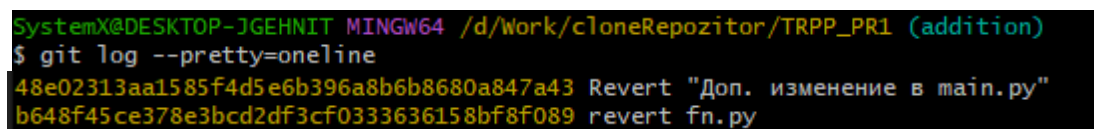
Рисунок 6.17 – Индексация изменений и добавление коммита



```
MINGW64:/d/Work/cloneRepozitor/TRPP_PR1
GNU nano 7.2 main.py
s = []
for i in range(1,101):
    if i%11==0:
        print(i," - делиться на 11")
```

Рисунок 6.18 – Revert 4-го коммита

Выполним команду `git revert HEAD` и произведём необходимые изменения.



```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/cloneRepozitor/TRPP_PR1 (addition)
$ git log --pretty=oneline
48e02313aa1585f4d5e6b396a8b6b8680a847a43 Revert "Доп. изменение в main.py"
b648f45ce378e3bcd2df3cf0333636158bf8f089 revert fn.py
```

Рисунок 6.19 – Выполнение revert над предпоследним коммитом

- 6) Выведем в консоли различия между веткой main и новой веткой.

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/cloneRepozitor/TRPP_PR1 (addition)
$ git diff main addition
diff --git a/fn.py b/fn.py
index 9bdf46f..f16c127 100644
--- a/fn.py
+++ b/fn.py
@@ -1,2 +1,2 @@
-for i in range(1, 16, 2):
-    print(i)
+for i in range(2, 17, 2):
+    print(i+1)
diff --git a/main.py b/main.py
index 50aa723..b6e61c5 100644
--- a/main.py
+++ b/main.py
@@ -1,2 +1,4 @@
-s = [1,2,3,4]
-print(s)
\ No newline at end of file
+s = []
+for i in range(1,101):
+    if i%11==0:
+        print(i," - делиться на 11")
\ No newline at end of file
diff --git a/readme.txt b/readme.txt
:...skipping...
diff --git a/fn.py b/fn.py
index 9bdf46f..f16c127 100644
--- a/fn.py
+++ b/fn.py
@@ -1,2 +1,2 @@
-for i in range(1, 16, 2):
-    print(i)
+for i in range(2, 17, 2):
+    print(i+1)
diff --git a/main.py b/main.py
index 50aa723..b6e61c5 100644
--- a/main.py
+++ b/main.py
@@ -1,2 +1,4 @@
-s = [1,2,3,4]
-print(s)
\ No newline at end of file
+s = []
+for i in range(1,101):
+    if i%11==0:
+        print(i," - делиться на 11")
\ No newline at end of file
diff --git a/readme.txt b/readme.txt
index 1769992..77b847c 100644
--- a/readme.txt
+++ b/readme.txt
@@ -1 +1 @@
-Два маленьких кода на Python
+Два маленьких обновлённых кода на Python
```

Рисунок 6.15 – Вывод различий между двумя ветками

- 7) Произведём слияние новой ветки с веткой main при помощи merge.

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/cloneRepozitor/TRPP_PR1 (addition)
$ git merge main
Already up to date.
```

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.

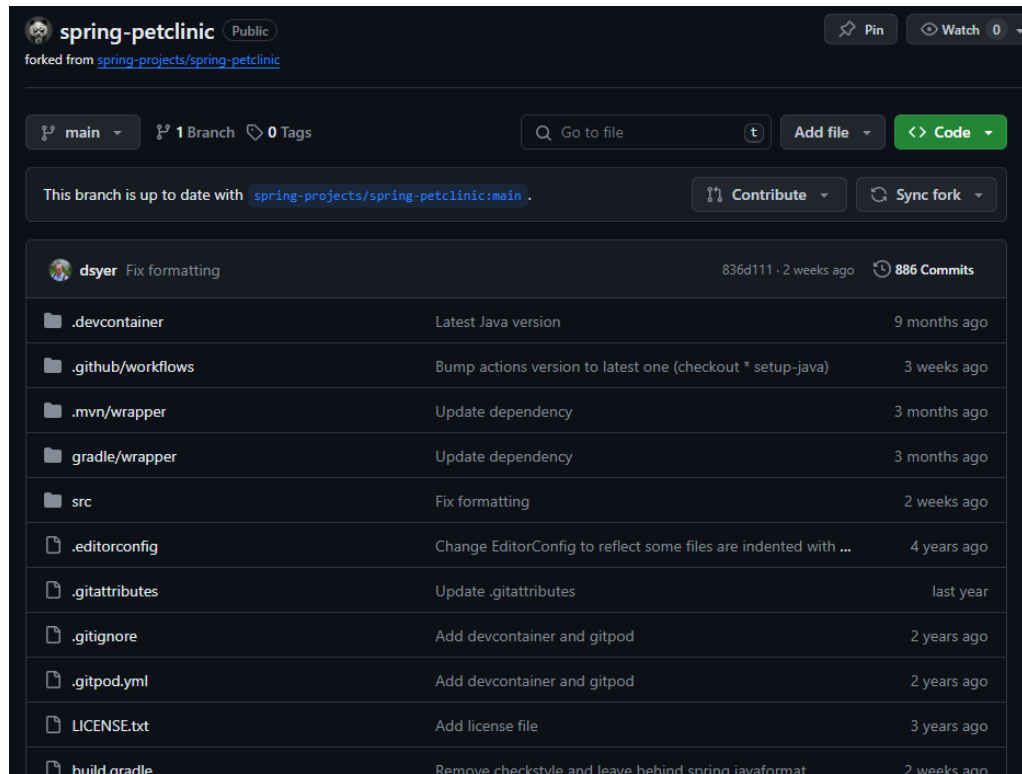


Рисунок 1.1 – Форк необходимого репозитория

2) Склонируем данный репозиторий на локальную машину.

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/forkRepozitor
$ git clone git@github.com:GrigorUN/spring-petclinic.git
Cloning into 'spring-petclinic'...
remote: Enumerating objects: 9664, done.
remote: Counting objects: 100% (29/29), done.
remote: Compressing objects: 100% (16/16), done.
remote: Total 9664 (delta 7), reused 17 (delta 5), pack-reused 9635
Receiving objects: 100% (9664/9664), 7.41 MiB | 4.47 MiB/s, done.
Resolving deltas: 100% (3604/3604), done.
```

Рисунок 1.2 – Копирование репозитория на локальную машину

3) Создадим две ветки branch1 и branch2.

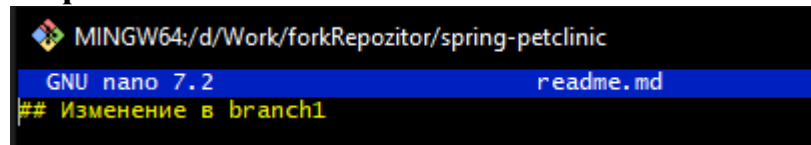
```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/forkRepozitor/spring-petclinic (main)
$ git branch branch1

SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/forkRepozitor/spring-petclinic (main)
$ git branch branch2
```

Рисунок 1.3 – Создание новых веток

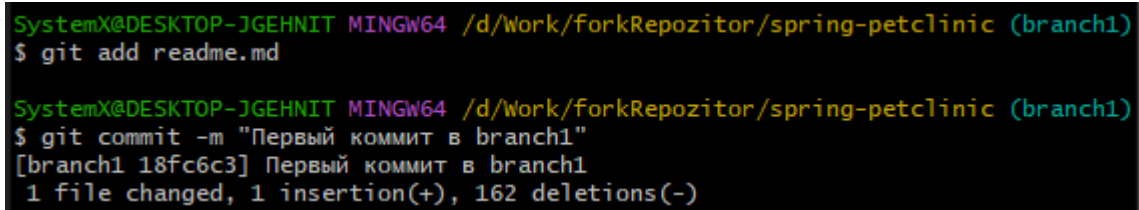


Проведём по три коммита в каждую из веток, которые меняют один и тот же кусочек файла.



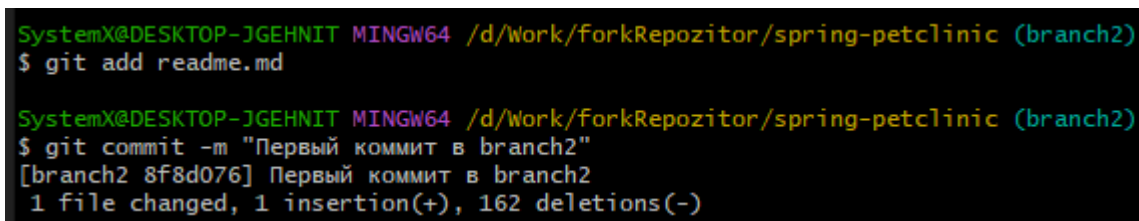
A screenshot of a terminal window showing the nano text editor. The title bar indicates the path is MINGW64:/d/Work/forkRepozitor/spring-petclinic. The editor is editing the file README.md. The current line is commented with '## Изменение в branch1'.

Рисунок 1.4 – Изменение в файле README.md



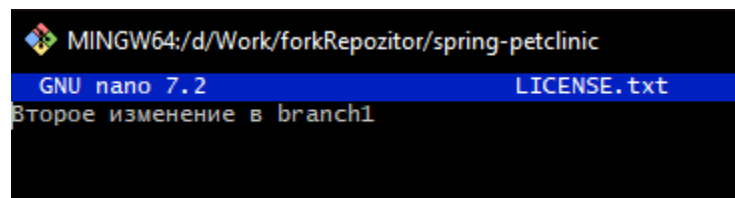
A terminal screenshot showing the execution of git commands in the directory /d/Work/forkRepozitor/spring-petclinic (branch1). The first command is 'git add README.md'. The second command is 'git commit -m "Первый коммит в branch1"', which results in a commit with hash 18fc6c3, indicating 1 file changed, 1 insertion(+), and 162 deletions(-).

Рисунок 1.5 – Первый коммит для первой ветки



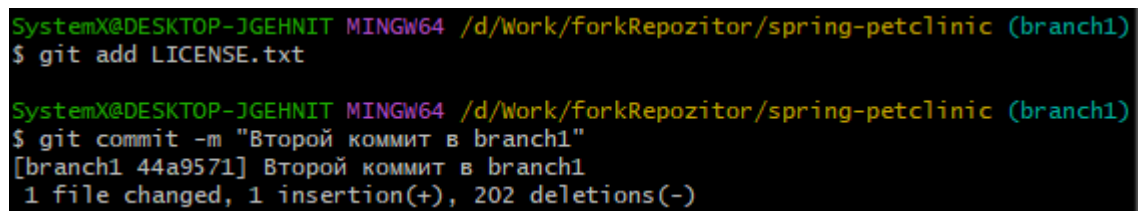
A terminal screenshot showing the execution of git commands in the directory /d/Work/forkRepozitor/spring-petclinic (branch2). The first command is 'git add README.md'. The second command is 'git commit -m "Первый коммит в branch2"', which results in a commit with hash 8f8d076, indicating 1 file changed, 1 insertion(+), and 162 deletions(-).

Рисунок 1.6 – Первый коммит для второй ветки



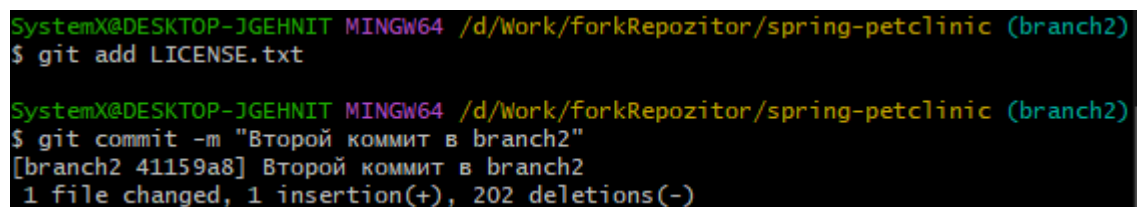
A screenshot of a terminal window showing the nano text editor. The title bar indicates the path is MINGW64:/d/Work/forkRepozitor/spring-petclinic. The editor is editing the file LICENSE.txt. The current line is commented with 'Второе изменение в branch1'.

Рисунок 1.7 – Изменение в файле LICENSE.txt



A terminal screenshot showing the execution of git commands in the directory /d/Work/forkRepozitor/spring-petclinic (branch1). The first command is 'git add LICENSE.txt'. The second command is 'git commit -m "Второй коммит в branch1"', which results in a commit with hash 44a9571, indicating 1 file changed, 1 insertion(+), and 202 deletions(-).

Рисунок 1.8 – Второй коммит для первой ветки



A terminal screenshot showing the execution of git commands in the directory /d/Work/forkRepozitor/spring-petclinic (branch2). The first command is 'git add LICENSE.txt'. The second command is 'git commit -m "Второй коммит в branch2"', which results in a commit with hash 41159a8, indicating 1 file changed, 1 insertion(+), and 202 deletions(-).

Рисунок 1.9 – Второй коммит для второй ветки

```
MINGW64:/d/Work/forkRepozitor/spring-petclinic
GNU nano 7.2 mvnw
Третье изменение в branch1
```

Рисунок 1.10 – Изменение в файле mvnw

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/forkRepozitor/spring-petclinic (branch1)
$ git add mvnw

SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/forkRepozitor/spring-petclinic (branch1)
$ git commit -m "Третий коммит в branch1"
[branch1 695637f] Третий коммит в branch1
1 file changed, 1 insertion(+), 308 deletions(-)
```

Рисунок 1.11 – Третий коммит для первой ветки

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/forkRepozitor/spring-petclinic (branch2)
$ git add mvnw

SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/forkRepozitor/spring-petclinic (branch2)
$ git commit -m "Третий коммит в branch2"
[branch2 ee2e63a] Третий коммит в branch2
1 file changed, 1 insertion(+), 308 deletions(-)
```

Рисунок 1.12 – Третий коммит для второй ветки

#### 4) Выполним слияние ветки branch1 в ветку branch2.

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/forkRepozitor/spring-petclinic (branch2)
$ git merge branch1
Auto-merging LICENSE.txt
CONFLICT (content): Merge conflict in LICENSE.txt
Auto-merging mvnw
CONFLICT (content): Merge conflict in mvnw
Auto-merging readme.md
CONFLICT (content): Merge conflict in readme.md
Automatic merge failed; fix conflicts and then commit the result.
```

Рисунок 1.13 – Ошибки, возникшие при слиянии

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

```
GNU nano 7.2 readme.md
<<<<<<< HEAD
## Изменения в branch2
=====
## Изменение в branch1
>>>>>> branch1
```

Рисунок 1.14 – Файл с метками, оставленными git

Отредактируем файлы и уберём метки, оставленные git.

```
GNU nano 7.2
## Изменения в branch2
```

Рисунок 1.15 – Отредактированное место в файле readme

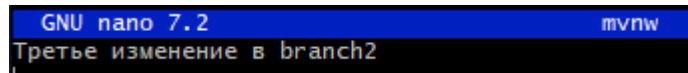


Рисунок 1.16 – Отредактированное место в файле mvnw

Сохраним данный файл и введём необходимые команды.

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/forkRepozitor/spring-petclinic (branch2|MERGEI
NG)
$ git add readme.md

SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/forkRepozitor/spring-petclinic (branch2|MERGEI
NG)
$ git add mvnw

SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/forkRepozitor/spring-petclinic (branch2|MERGEI
NG)
$ git add LICENSE.txt

SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/forkRepozitor/spring-petclinic (branch2|MERGEI
NG)
$ git commit -m "merged branch1 and branch2"
[branch2 b5a3481] merged branch1 and branch2
```

Рисунок 1.17 – Необходимые для разрешения конфликта команды\

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/forkRepozitor/spring-petclinic (branch2)
$ git merge branch1
Already up to date.
```

Рисунок 1.18 – Успешное слияние после исправления ошибок

## 5) Выгрузим все изменения во всех ветках в удалённый репозиторий.

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/forkRepozitor/spring-petclinic (branch2)
$ git push origin branch1
Enumerating objects: 13, done.
Counting objects: 100% (13/13), done.
Delta compression using up to 12 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (9/9), 972 bytes | 972.00 KiB/s, done.
Total 9 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), completed with 1 local object.
remote:
remote: Create a pull request for 'branch1' on GitHub by visiting:
remote:   https://github.com/GrigorUN/spring-petclinic/pull/new/branch1
remote:
To github.com:GrigorUN/spring-petclinic.git
 * [new branch]      branch1 -> branch1

SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/forkRepozitor/spring-petclinic (branch2)
$ git push origin branch2
Enumerating objects: 20, done.
Counting objects: 100% (20/20), done.
Delta compression using up to 12 threads
Compressing objects: 100% (8/8), done.
Writing objects: 100% (12/12), 1.24 KiB | 1.24 MiB/s, done.
Total 12 (delta 4), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (4/4), completed with 1 local object.
remote:
remote: Create a pull request for 'branch2' on GitHub by visiting:
remote:   https://github.com/GrigorUN/spring-petclinic/pull/new/branch2
remote:
To github.com:GrigorUN/spring-petclinic.git
 * [new branch]      branch2 -> branch2
```

Рисунок 1.19 – Выгрузка изменений в удалённый репозиторий

6) Проведём ещё три коммита в ветку **branch1**.

```
GNU nano 7.2
## Изменение в branch1
# Дополнительный коммит
```

Рисунок 1.20 – Изменение в файле readme.md

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/forkRepozitor/spring-petclinic (branch1)
$ git add readme.md

SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/forkRepozitor/spring-petclinic (branch1)
$ git commit -m "Первый"
[branch1 9ce896d] Первый
1 file changed, 1 insertion(+)
```

Рисунок 1.21 – Первый новый коммит

```
GNU nano 7.2
Второе изменение в branch1
Дополнительное изменение|
```

Рисунок 1.22 – Изменение в файле LICENSE.txt

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/forkRepozitor/spring-petclinic (branch1)
$ git add LICENSE.txt

SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/forkRepozitor/spring-petclinic (branch1)
$ git commit -m "Второй"
[branch1 84951bf] Второй
1 file changed, 2 insertions(+), 1 deletion(-)
```

Рисунок 1.22 – Второй новый коммит

```
GNU nano 7.2
Третье изменение в branch1
Дополнительное изменение
```

Рисунок 1.23 – Изменение в файле mvnw

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/forkRepozitor/spring-petclinic (branch1)
$ git add mvnw

SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/forkRepozitor/spring-petclinic (branch1)
$ git commit -m "Третий"
[branch1 2547b1a] Третий
1 file changed, 1 insertion(+)
```

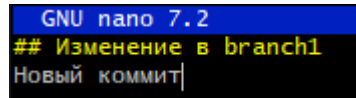
Рисунок 1.24 – Третий новый коммит

7) Склонируем репозиторий ещё раз в другую директорию.

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/forkRepozitor2
$ git clone git@github.com:GrigorUN/spring-petclinic.git
Cloning into 'spring-petclinic'...
remote: Enumerating objects: 9685, done.
remote: Counting objects: 100% (50/50), done.
remote: Compressing objects: 100% (24/24), done.
remote: Total 9685 (delta 15), reused 36 (delta 11), pack-reused 9635
Receiving objects: 100% (9685/9685), 7.41 MiB | 1.10 MiB/s, done.
Resolving deltas: 100% (3612/3612), done.
```

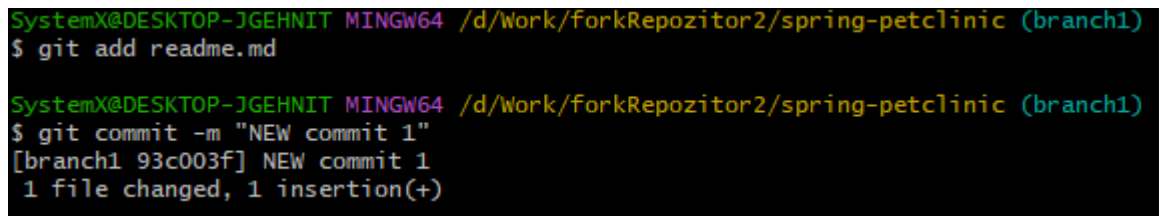
Рисунок 1.25 – Копирование репозитория в другую директорию

8) В новом клоне репозитория сделаем три коммита в ветке **branch1**.



```
GNU nano 7.2
## Изменение в branch1
Новый коммит|
```

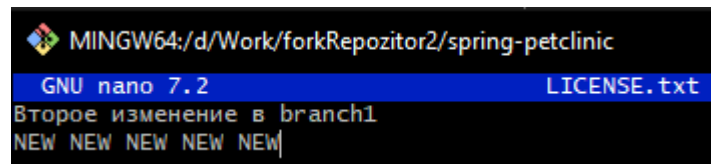
Рисунок 1.26 – Изменение в файле README.md



```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/forkRepozitor2/spring-petclinic (branch1)
$ git add readme.md

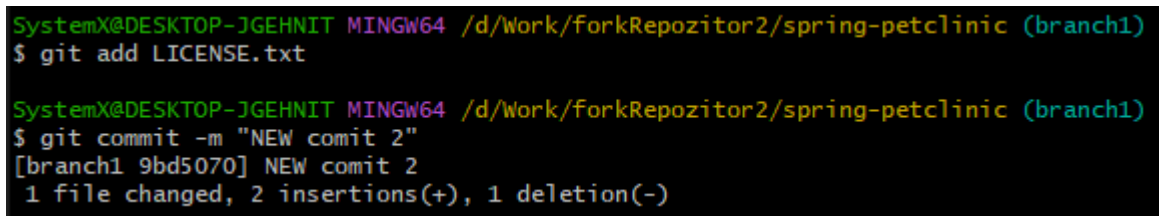
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/forkRepozitor2/spring-petclinic (branch1)
$ git commit -m "NEW commit 1"
[branch1 93c003f] NEW commit 1
1 file changed, 1 insertion(+)
```

Рисунок 1.27 – Первый коммит в новой копии репозитория



```
MINGW64:/d/Work/forkRepozitor2/spring-petclinic
GNU nano 7.2 LICENSE.txt
Второе изменение в branch1
NEW NEW NEW NEW NEW|
```

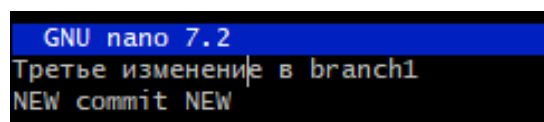
Рисунок 1.28 – Изменение в файле LICENSE.txt



```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/forkRepozitor2/spring-petclinic (branch1)
$ git add LICENSE.txt

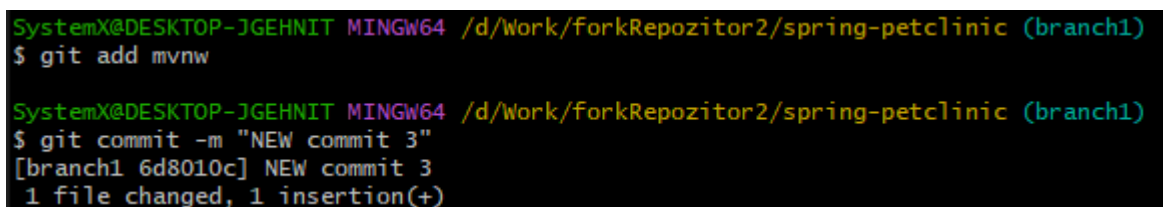
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/forkRepozitor2/spring-petclinic (branch1)
$ git commit -m "NEW comit 2"
[branch1 9bd5070] NEW comit 2
1 file changed, 2 insertions(+), 1 deletion(-)
```

Рисунок 1.29 – Второй коммит в новой копии репозитория



```
GNU nano 7.2
Третье изменение в branch1
NEW commit NEW|
```

Рисунок 1.30 – Изменение в файле mvnw



```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/forkRepozitor2/spring-petclinic (branch1)
$ git add mvnw

SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/forkRepozitor2/spring-petclinic (branch1)
$ git commit -m "NEW commit 3"
[branch1 6d8010c] NEW commit 3
1 file changed, 1 insertion(+)
```

Рисунок 1.31 – Третий коммит в новой копии репозитория

9) **Выгрузим все изменения из нового репозитория в удалённый репозиторий.**

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/forkRepozitor2/spring-petclinic (branch1)
$ git push origin branch1
Enumerating objects: 13, done.
Counting objects: 100% (13/13), done.
Delta compression using up to 12 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (9/9), 906 bytes | 906.00 KiB/s, done.
Total 9 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), completed with 1 local object.
To github.com:GrigorUN/spring-petclinic.git
 695637f..6d8010c branch1 -> branch1
```

Рисунок 1.32 – Выгрузка изменений из нового репозитория в удалённый

10) **Вернёмся в старый клон репозитория и выгрузим изменения с опцией –force.**

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/forkRepozitor/spring-petclinic (branch1)
$ git push origin --force branch1
Enumerating objects: 13, done.
Counting objects: 100% (13/13), done.
Delta compression using up to 12 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (9/9), 988 bytes | 988.00 KiB/s, done.
Total 9 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), completed with 1 local object.
To github.com:GrigorUN/spring-petclinic.git
+ 6d8010c...2547b1a branch1 -> branch1 (forced update)
```

Рисунок 1.33 – Выгрузка изменений с опцией –force

11) **Получим все изменения в новом репозитории.**

```
SystemX@DESKTOP-JGEHNIT MINGW64 /d/Work/forkRepozitor2/spring-petclinic (branch1)
$ git pull
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 9 (delta 3), reused 9 (delta 3), pack-reused 0
Unpacking objects: 100% (9/9), 968 bytes | 3.00 KiB/s, done.
From github.com:GrigorUN/spring-petclinic
+ 6d8010c...2547b1a branch1 -> origin/branch1 (forced update)
```

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.

2. 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.

3. 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.

4. 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.

5. 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.

6. 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.

7. 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.

8. 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.