

# Практическая работа №1. Git и Github.

## Теория.

### *Что такое Git?*

*Git* — это распределённая система контроля версий, которая помогает отслеживать изменения в исходном коде или других файлах в проекте. Она позволяет работать как в одиночку, так и в команде, предоставляя инструменты для сохранения, отката и совместной работы над проектами.

- История изменений: *Git* хранит всю историю изменений в виде коммитов, что позволяет в любой момент вернуться к предыдущим версиям файла или всей системы.
- Распределённая структура: Каждый разработчик работает с локальной копией репозитория, что позволяет работать автономно, а затем синхронизироваться с основной версией.
- Ветвление и слияние: *Git* позволяет легко создавать ветки для разработки новых функций или исправления багов, не влияя на основную версию проекта. Позже ветки можно объединить.
- Конфликты и их разрешение: При слиянии изменений *Git* помогает выявить и разрешить конфликты, возникающие, если два разработчика изменили один и тот же файл.
- Вовлеченность всей команды: При попытке слияния вашей ветки как отдельного разработчика с главной веткой проекта всей команды, требуется подтверждение всех членов и участников команды.

## Структура.

Структуру Git можно представить, как дерево с ветками, узлами и файлами. Вот основные элементы структуры Git:

- Локальный репозиторий;
- Удаленный репозиторий;
- Коммиты;
- Ветки;
- Индекс;

Итак, разберемся как все устроено.

В начале нам требуется создать репозиторий. Он может быть, как локальным, находящейся только на вашем компьютере, без возможности клонирования другими разработчиками, так и удалённым, находящийся на хостинге Git Hub.

*Что нам это дает?*

За этим репозиторием можно установить наблюдение, с целью отслеживания всех изменений. А именно:

- Создание и удаление файлов.
- Изменение содержимого файлов.

При создании репозитория по умолчанию существует ветка *main* или *master*.

Она является главной веткой вашего репозитория и в ней будет лежать самая стабильная и правильная версия вашего проекта.

Но по мимо главные ветки, дабы не попортить код, например, при добавление новой функции, разработчикам будет правильно создать отдельную ветку, которая будет является клоном главной ветки или той ветки, которая является самой стабильной.

После того, как программист работающий в второстепенной ветке, доделает новую функцию, протестирует ее, он может сделать слияние этой ветки и главной ветки проекта. Об этом мы поговорим чуть позже.

Наличие веток, помогает программистам обособиться от главной версии проекта, дабы в полной мере начать писать новый код, без возможности испортить предыдущий.

Коммиты. Они представляют собой зафиксированные изменения в репозитории.

Структуру коммитов можно представить так:

Есть репозиторий, от него идет наша главная ветка *main-master*. У каждой ветки может быть бесчисленное число коммитов (наших зафиксированных изменений.)

Но и это еще не все. От самих коммитов, мы так же можем создавать ветки и уже в них фиксировать новые коммиты.

У нас есть две ветки: *Ветка Main* и *Ветка N*. *Ветка N* клонирована от *Ветки Main*. Они обе имеют отдельные коммиты (A, B, C, D, F, E). Далее происходит слияние двух веток (коммиты: E в C). После слияния в ветку *main* добавляются все изменения ветки *N*, без потери содержимого Ветки *main*, так как создается отдельный коммит. (Рисунок 1)

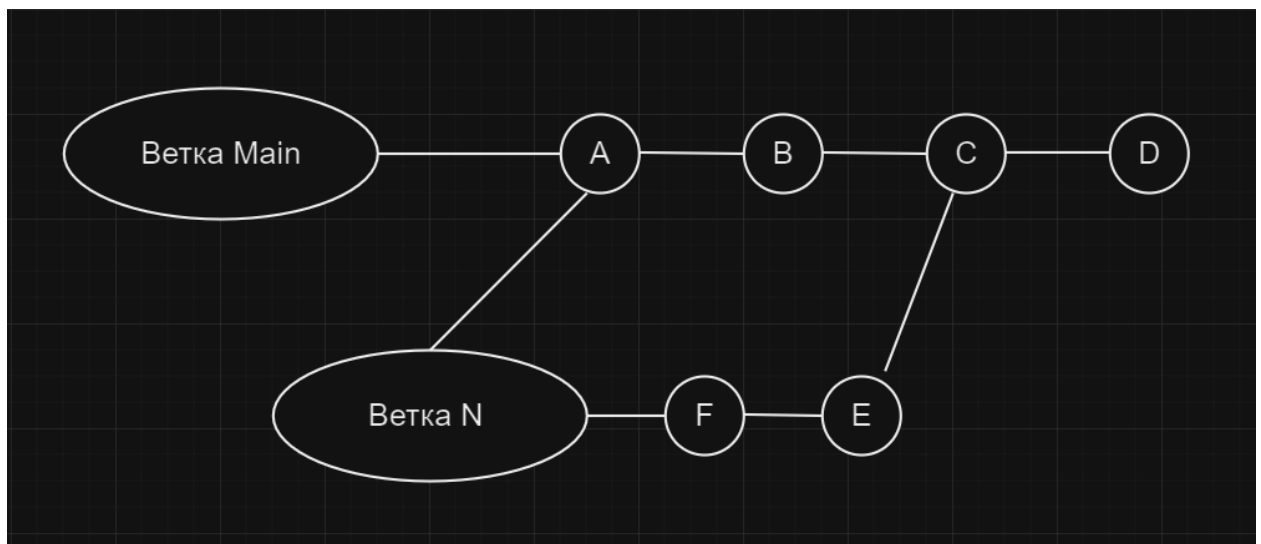


Рисунок 1 – Структура Веток и Коммитов

## *Список основных команд.*

Настройки Git:

- `git config --global user.name "Ваше имя"`
- `git config --global user.email "Ваше Email"`

Работа с репозиторием:

- `git init`
- `cd "Нужная папка".`
- `git remote add origin <Ваш url репозитория>`
- `git clone <Ваш url репозитория>`
- `git pull origin <Нужная ветка>`

Работа с Коммитами:

- `git add <указать название нужных файлов>`
- `git add .`
- `git commit - m "Ваше сообщение для название коммита"`
- `git log -`
- `git checkout <хеш нужного коммита>;`

Работа с ветками:

- `git push origin <Имя ветки, которую хотим загрузить>`
- `git branch`
- `git branch <Имя ветки>`
- `git checkout <Имя ветки>`
- `git merge <Имя ветки из которой хотим взять изменения>`
- `git rebase <Имя ветки из которой хотим взять изменения>`

## Практическая часть.

### *Установка, настройка Window 10 и сопутствующих программ.*

Для того, чтобы нам перейти к изучению Git(консоль), нам с вами требуется установить и запустить образ Windows 10, для этого:

- Запустите VirtualBoxes.
- Используя образ Window 10, создайте виртуальную машину и настройте ее.

После того как виртуальная машина будет настроена, мы с вами переходим в работе с Git.

Для этого переходим на официальный сайт <https://git-scm.com/downloads>.

Скачиваем и устанавливаем Git на свою виртуальную машину.

Как все будет готово, перейдем на официальный сайт <https://github.com/>, проходим регистрацию, чтобы в последствии загружать наш проект в отдельный удаленный репозиторий.

После того как установили Git и зарегистрировались на GitHub, перейдем к настройке Git на своей виртуальной машине.

## Настройка и работа с Git(консоль).

Для этого, создадим в удобном для нас месте папку, за которой мы будем «Следить» и перейдем в нее.

В ней нажмем правую кнопку мыши и увидим строчку “*Open Git bash here*”, нажимаем на неё и нам отрывается окно консоли Git. (Рисунок 2,3)

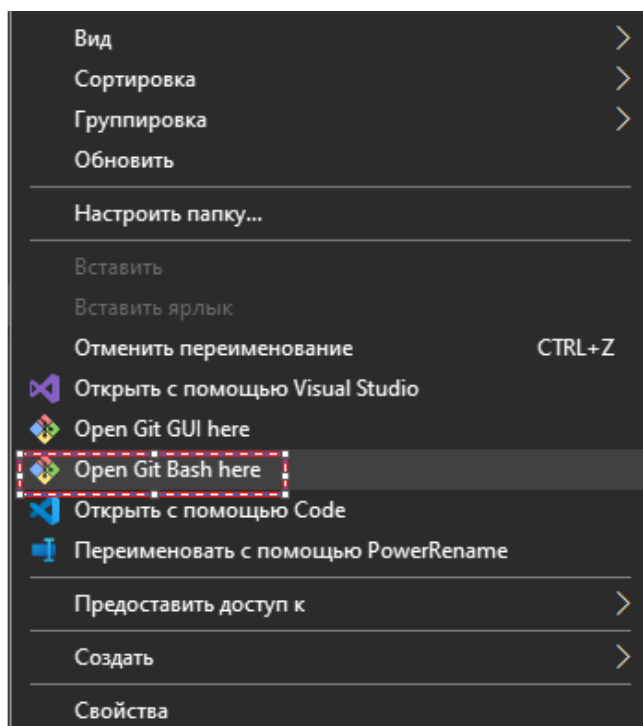


Рисунок 2 – Запуск консоли Git

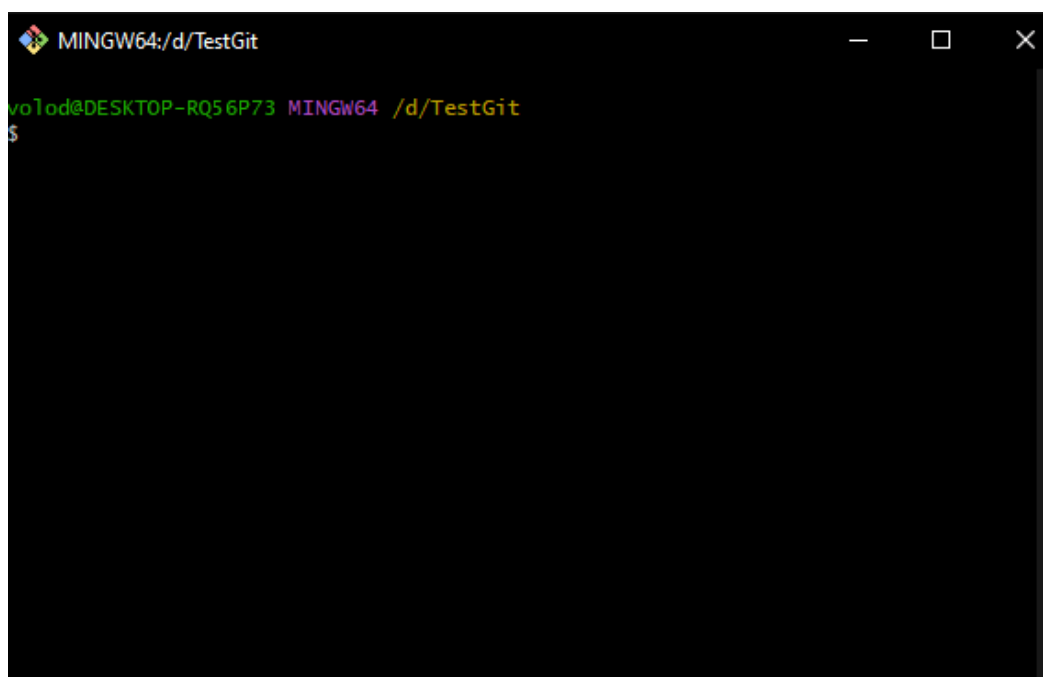


Рисунок 3 – Окно консоли Git

### *Примечание:*

Git работает с любым представлением командной строки. (Рисунок 4)

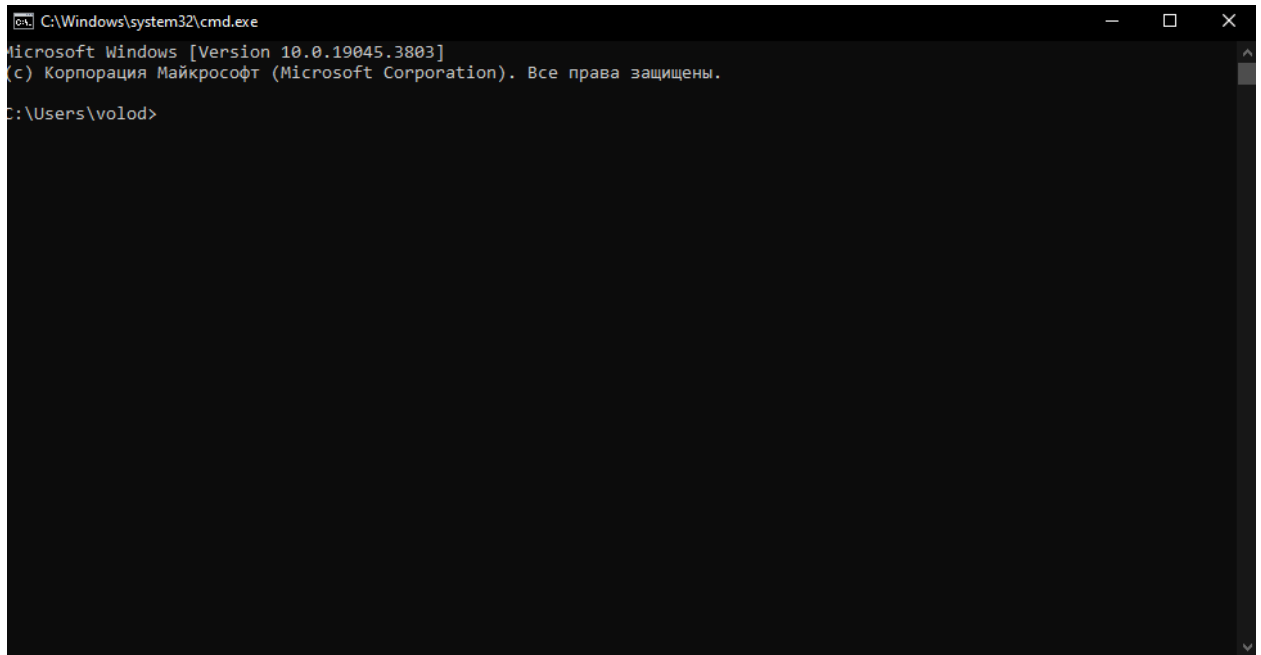


Рисунок 4 – Окно CMD

Настройка Git заключается в трех командах:

- Установка Имени для маркировки изменений(Коммитов).

```
git config --global user.name "Ваше имя"
```

- Установка Email для маркировки изменений(Коммитов).

```
git config --global user.email "Ваше Email"
```

- Установка Локального репозитория.

```
git init
```

### *Примечание*

Две первые команды, подписывают все коммиты которые создали вашим именем. Из-за приставки global, данные настройки Git применяются ко всем файлам на компьютере.

Итак, теперь перейдем к тому, чтобы превратить нужную нам папку в репозиторий за которым Git будет следить.

Первое, что надо сделать – это перейти в нужную нам папку.

Так как с помощью правой кнопкой мыши мы запустили Git(Консоль) в той папке, которую мы хотим отслеживать, нам и не надо в нее переходить

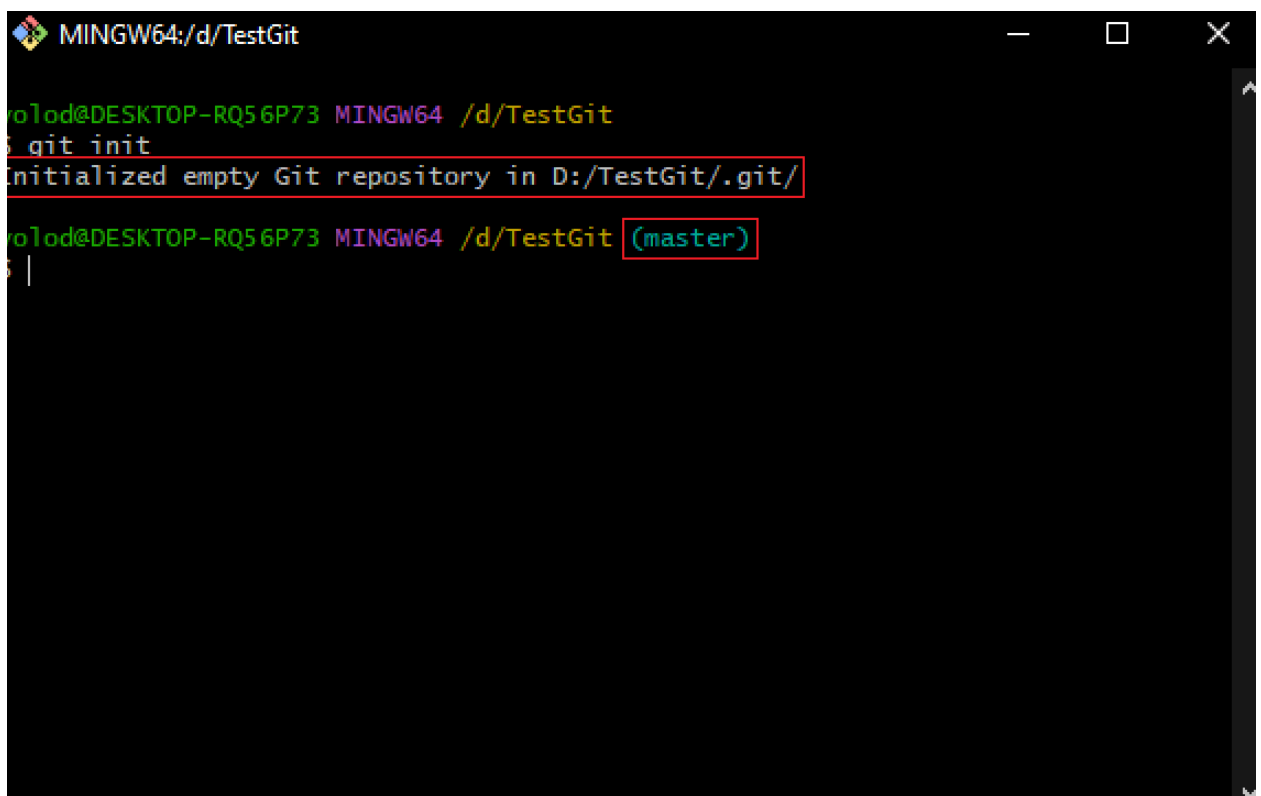
### *Примечание:*

Для того, чтобы перейти в папку, используем команду:  
`cd "Нужная папка"`.

### *Работа с репозиториями.*

После того, как перешли в нужную папку (чуть ранее мы ее создали), нужно выполнить команду, того, что бы Git начал «Слежку» за папкой.

Для этого воспользуемся командой: `git init`, после чего мы увидим сообщение от Git. (Рисунок 5)



```
MINGW64:/d/TestGit
olod@DESKTOP-RQ56P73 MINGW64 /d/TestGit
$ git init
initialized empty Git repository in D:/TestGit/.git/
olod@DESKTOP-RQ56P73 MINGW64 /d/TestGit (master)
$ |
```

Рисунок 5 – Установка репозитория

Обратите внимание, что следующая строчка, после ответа от Git, показывает нам в какой ветке мы находимся (по умолчанию ветка называется Master).



Реализация локального репозитория окончена. Теперь переходим к созданию удаленного репозитория.

Чуть ранее вы зарегистрировались на GitHub. Перейдем на сайт и создадим удаленный репозиторий, для того что бы опубликовать наш локальный репозиторий. (Рисунок 6-8)

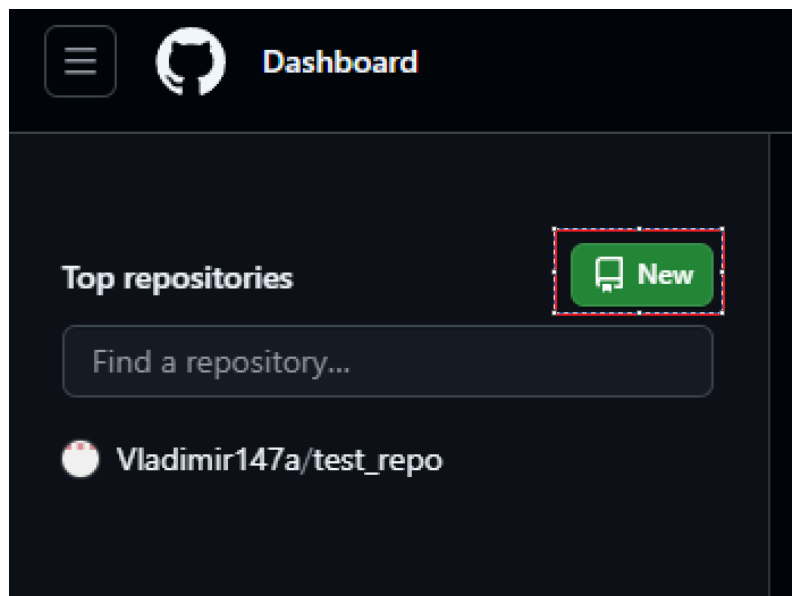


Рисунок 6 – Создание нового репозитория

Дайте репозиторию имя, укажите степень приватности.(Рисунок 7)

The image shows the 'Create a new repository' form on GitHub. The form has a title 'Create a new repository' and a subtitle 'A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)'. Below this, a note states 'Required fields are marked with an asterisk (\*)'. The form contains several fields: 'Owner' (a dropdown menu showing 'Vladimir147a'), 'Repository name' (a text input field containing 'Test', highlighted with a red dashed box), 'Description' (an optional text input field), 'Public/Private' (radio buttons, with 'Public' selected and highlighted by a red dashed box), 'Initialize this repository with:' (checkboxes for 'Add a README file' and 'Add .gitignore'), 'Choose a license' (a dropdown menu showing 'License: None'), and a 'Create repository' button at the bottom right, which is also highlighted with a red dashed box. A note at the bottom states 'You are creating a public repository in your personal account.'

Рисунок 7 – Создание нового репозитория

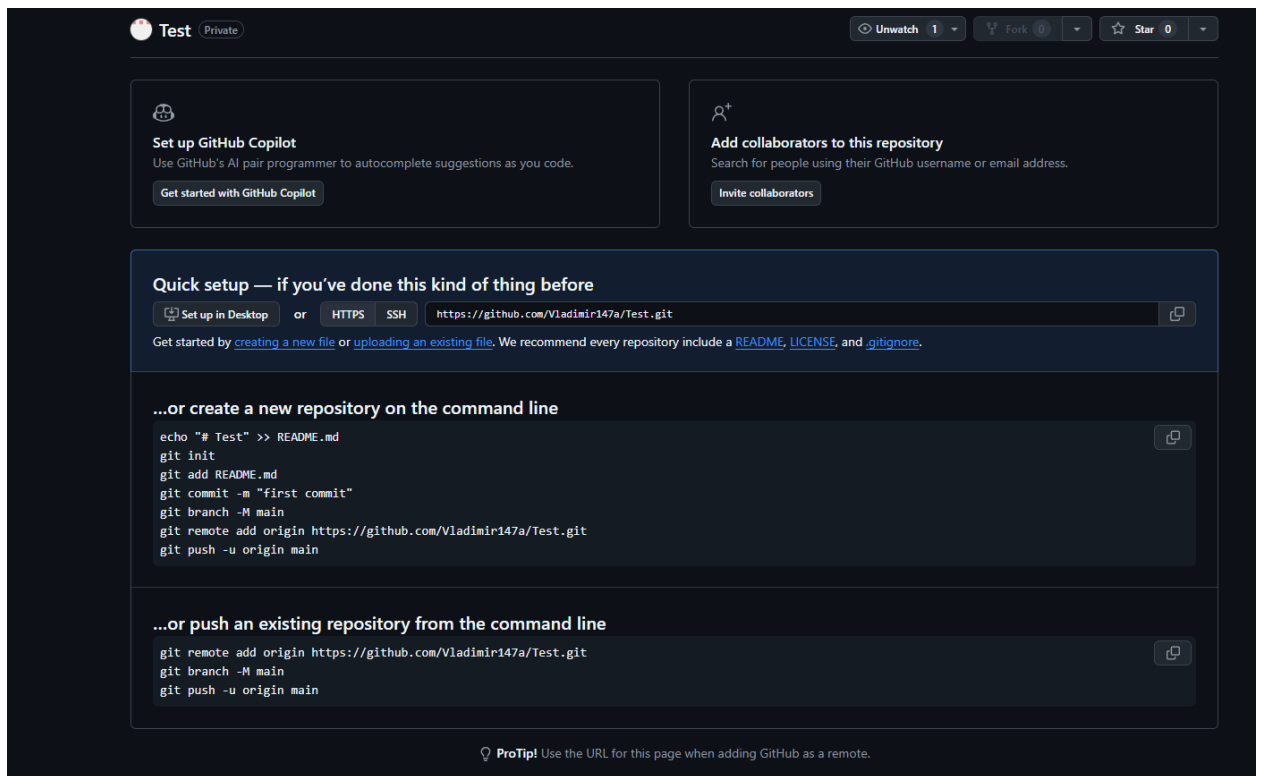


Рисунок 8 – Окно созданного репозитория

Так, теперь перейдем к моменту связывания нашего локального репозитория с удаленным. Для этого используем команду:

```
git remote add origin <Ваш url репозитория>
```

*Примечание:*

Чтобы взять ваш Url, нужно перейти на GitHub, выбрать нужный репозиторий с которым хотите связать и скопировать его Url. (Рисунок 9)

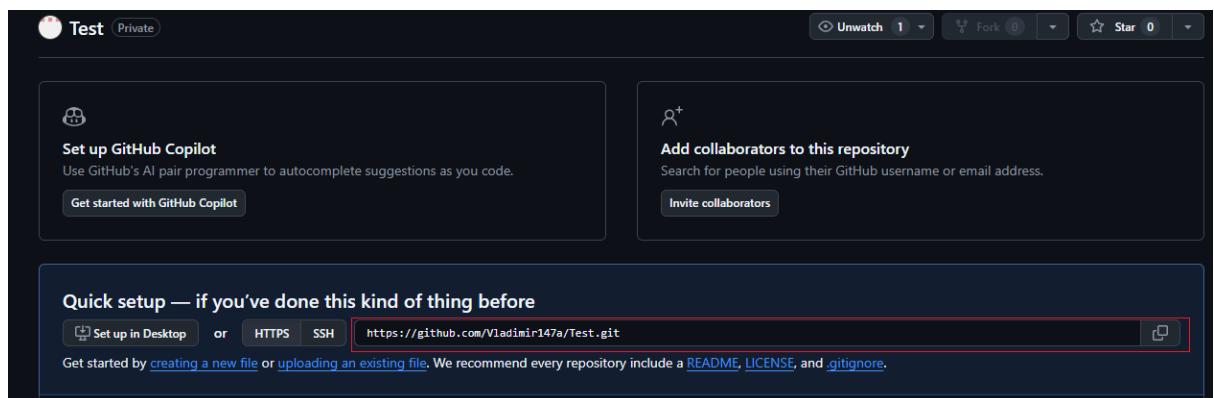


Рисунок 9 – Url удаленного репозитория

После успешного выполнения этой команды, наши два репозитория будут связаны.

### *Примечание:*

Даже если эти два репозитория связаны, без специальной команды, в удаленный репозиторий ничего не будет сохраняться.

Далее мы должны рассмотреть возможность выгрузки с удаленного репозитория файлов на локальный. Переходим в нужную нам папку и используем команду:

```
git clone <Ваш url репозитория>
```

Теперь, в наш новую локальную папку с клонировался удаленный репозиторий. При этом, удаленный репозиторий также связался с этой папкой, так как, она теперь является локальным репозиторием. (Рисунок 10)

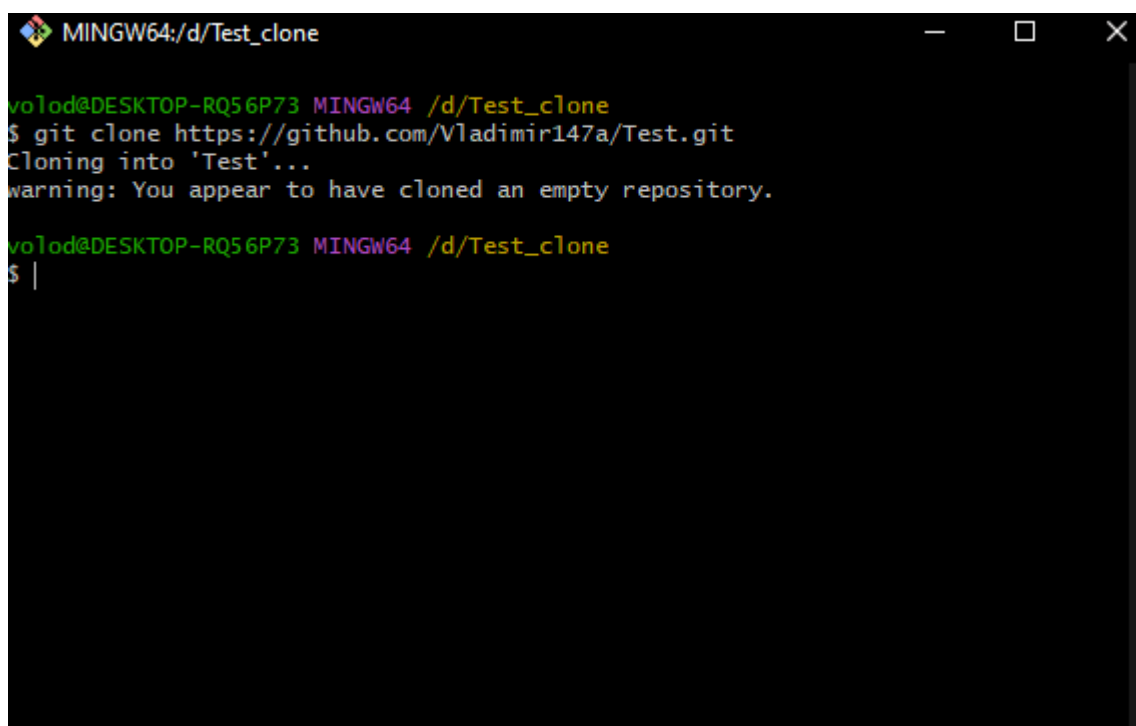
A screenshot of a terminal window titled 'MINGW64:/d/Test\_clone'. The prompt is 'volod@DESKTOP-RQ56P73 MINGW64 /d/Test\_clone'. The user enters the command '\$ git clone https://github.com/Vladimir147a/Test.git'. The terminal output shows 'Cloning into 'Test'...' followed by a warning: 'warning: You appear to have cloned an empty repository.' The prompt returns to '\$ |'.

Рисунок 10 – Клонирование репозитория

После применения команды, в нашей папке появилась еще одна папка. Она и является клоном нашего удаленного репозитория. (Рисунок 11)

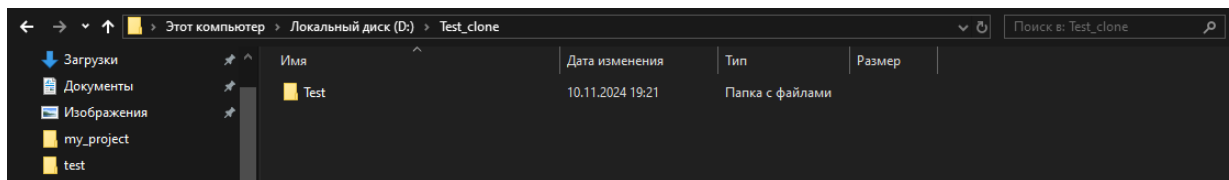


Рисунок 12 – Клонированная папка

### *Примечание:*

При указание того, что выбранная папка является репозиторием, внутри нее создается системная папка с название «.git».

Мы успешно все связали, и теперь можем перейти к созданию коммитов и отслеживание изменений в главной ветке *Master*.

Для того, чтобы отслеживать изменения, создадим текстовый документ в локальном репозитории и поместим в него какой-нибудь текст.

Итак, для того, чтобы понять в каком статусе находятся файлы внутри локального репозитория, мы используем команду:

```
git status
```

После того, как мы применили команду, Git выделит красным цветом, названия тех файлов, изменения которых, не были сохранены. (Рисунок 13, 14)

```
volod@DESKTOP-RQ56P73 MINGW64 /d/TestGit (master)
$ git status
On branch master

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)
volod@DESKTOP-RQ56P73 MINGW64 /d/TestGit (master)
```

Рисунок 13 – Ответ Git

Имя	Дата изменения	Тип	Размер
.git	10.11.2024 19:36	Папка с файлами	
test	10.11.2024 19:36	Текстовый докум...	1 КБ

Рисунок 13 –Набор файлов репозитории

Этим ответом, Git, показывает, что он знает, что в репозитории что-то изменилось, но при этом пока он это не сохранял.

Для того, чтобы сказать гиту, что файлы, которые он пометил надо сохранить, а в последствии сделать коммит этих файлов, требуется прописать команду:

`git add <указать название нужных файлов>`

или

`git add .` – точка показывает то, что мы хотим выбрать все файлы

После того, как мы применим эту команду, при повторном использовании команды `git status` Git пометит эти файлы зеленым цветом, то есть покажет какие файлы сохранит в коммит (Рисунок 14).

```

volod@DESKTOP-RQ56P73 MINGW64 /d/TestGit (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   test.txt

volod@DESKTOP-RQ56P73 MINGW64 /d/TestGit (master)

```

Рисунок 14 – Применение изменений

## *Примечание*

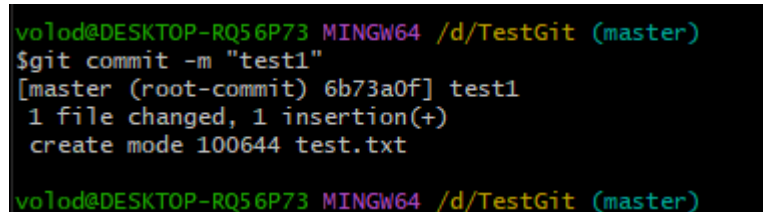
Если ваш проект был кем-то обновлён, и вы хотите обновить свой локальный репозиторий без его непосредственного клонирования, то можно использовать команду (`git pull origin <Нужная ветка>`). После применения данной команды, ваша выбранная ветка на локальном репозитории, обновится и в нее добавятся все коммиты, которых не было. Сама ветка станет самой последней версией.

## *Работа с коммитами.*

Ну и после этого, нам требуется сохранить наши изменения или же сделать коммит, командой:

```
git commit -m "Ваше сообщение для название коммита"
```

После успешного выполнения команды, Git пришлет сообщение о том, что все прошло успешно. (Рисунок 15)



```
volod@DESKTOP-RQ56P73 MINGW64 /d/TestGit (master)
$git commit -m "test1"
[master (root-commit) 6b73a0f] test1
1 file changed, 1 insertion(+)
create mode 100644 test.txt
volod@DESKTOP-RQ56P73 MINGW64 /d/TestGit (master)
```

Рисунок 15 – Создание коммитов

Теперь, если мы изменим файл, а после этого захотим вернуться к первоначальному виду файла, то мы сможем откатиться с помощью ранее созданного коммита.

Чтобы проверить это, изменим файл, создадим новый коммит и попытаемся вернуть файл к первоначальному виду.

Это задание сделайте самостоятельно.

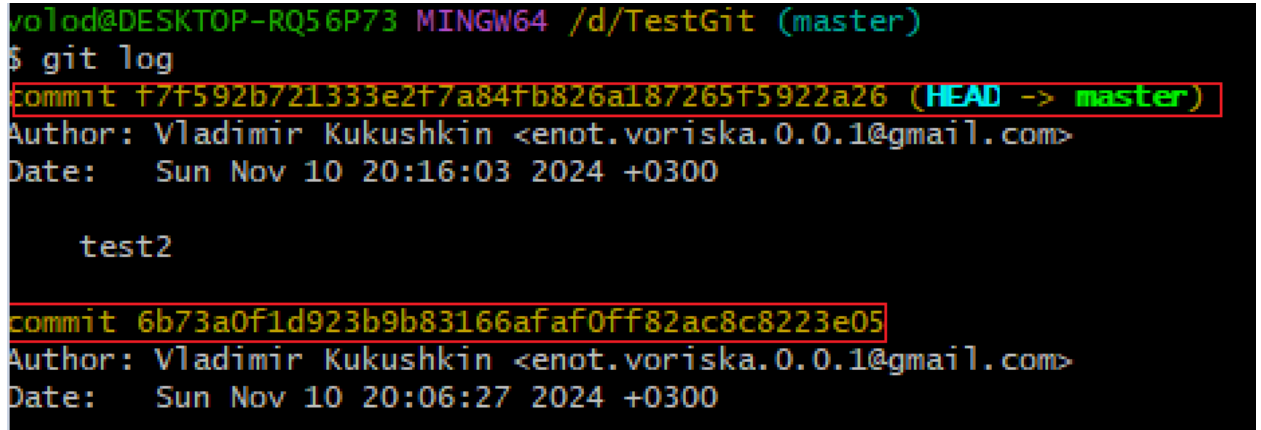
После того, как создали новый коммит с изменения, попытаемся вернуть предыдущий коммит, с помощью команд:

```
git log – вернет список всех коммитов;
```

```
git checkout <хеш нужного коммита>;
```

### *Примечание:*

Хеш коммита можно узнать из списка коммитов, при вызове команды `git log`. (Рисунок 16)



```
vo1od@DESKTOP-RQ56P73 MINGW64 /d/TestGit (master)
$ git log
commit f7f592b721333e2f7a84fb826a187265f5922a26 (HEAD -> master)
Author: Vladimir Kukushkin <enot.voriska.0.0.1@gmail.com>
Date: Sun Nov 10 20:16:03 2024 +0300

    test2

commit 6b73a0f1d923b9b83166afaf0ff82ac8c8223e05
Author: Vladimir Kukushkin <enot.voriska.0.0.1@gmail.com>
Date: Sun Nov 10 20:06:27 2024 +0300
```

Рисунок 16 – Хеши коммитов

Итак, после применение первого коммита, текст внутри файла поменялся.

Так, все эти коммиты пока хранятся только на локальном репозитории, чтобы их переместить в удаленный, нам нужно использовать следующую команду:

```
git push origin <Имя ветки, которую хотим загрузить>
```

### *Примечание*

Слово Origin означает, имя вашего удаленного репозитория, который мы подключали еще в самом начале.

Итак, перейдем на сайт GitHub и посмотрим загрузились ли файлы. (Рисунок 17)

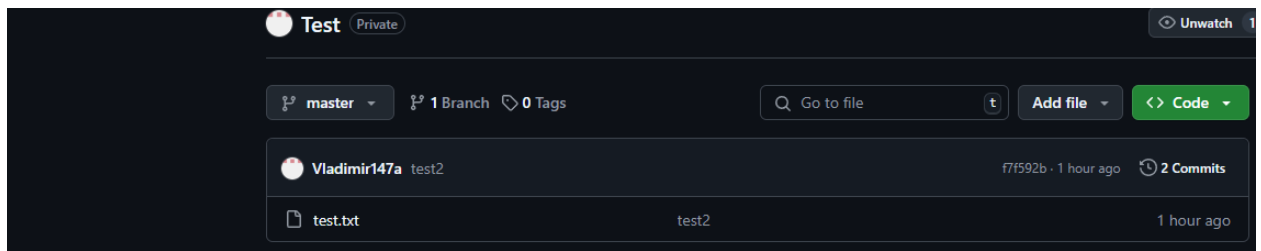


Рисунок 17 – Результат загрузки

Чуть ранее, мы смотрели как клонировать проект, теперь же мы рассмотрим момент, как обновить целую ветку, например, другому разработчику.

С помощью команды `git pull origin <Имя ветки>` мы добавим в локальный репозиторий другого разработчика, все изменения, которые есть в удаленном.

### *Ветки.*

Итак, чтобы разграничить работу программистов в одном проекте, чтобы программисты не мешали друг другу и не портили общий код, существует такой объект как *Ветка*.

Ветка представляет собой, набор коммитов, который идет от какой-либо ветки.

То есть, у нас есть главная ветка *Master*. Мы захотели создать функцию, но, чтобы нормально прописать это функцию, мы с последнего коммита в ветке *Master*, делаем разветвление и создаем новую ветку, которая будет иметь свои коммиты и не как не повлияет на файлы внутри ветки *Master*.

После того, как мы все доделали в этой отдельной ветке, мы можем все изменения переместить в главную.

### *Приступим к созданию веток.*

Для начала нам нужно узнать, какие ветки существуют в нашем репозитории. Узнать это можно, с помощью команды:

```
git branch
```



После использования этой команды мы получаем список всех существующих веток. Звездочкой отмечается та ветка, в которой мы сейчас находимся. (Рисунок 18)

```
volod@DESKTOP-RQ56P73 MINGW64 /d/TestGit ((6b73a0f...))
$ git branch
* (HEAD detached at 6b73a0f)
  master

volod@DESKTOP-RQ56P73 MINGW64 /d/TestGit ((6b73a0f...))
$ |
```

Рисунок 18 – Список всех веток

Сейчас у нас есть всего одна главная ветка. Для того, что создать ветку нам нужно использовать команду:

```
git branch <Имя ветки>
```

После того, как мы выполнили команду создания ветки и запросили список веток, мы можем заметить, что список поменялся и в него добавилась новая ветка. (Рисунок 19)

```
volod@DESKTOP-RQ56P73 MINGW64 /d/TestGit ((6b73a0f...))
$ git branch
* (HEAD detached at 6b73a0f)
  master
  test_branch

volod@DESKTOP-RQ56P73 MINGW64 /d/TestGit ((6b73a0f...))
$ |
```

Рисунок 19 – Обновлённый список всех веток

Чтобы перейти на другую ветку, используем команду (Рисунок 19):

```
git checkout <Имя ветки>
```

```
volod@DESKTOP-RQ56P73 MINGW64 /d/TestGit ((6b73a0f...))
$ git checkout test_branch
Switched to branch 'test_branch'

volod@DESKTOP-RQ56P73 MINGW64 /d/TestGit (test_branch)
$ |
```

Рисунок 20 – Переход на другие ветки

Теперь, когда мы изменим или добавим файлы, создадим на основе этих изменения коммиты, они будут сохраняться в этой ветке. А при переходе на

другую ветку, у нас изменяется все файлы, на последние версии файла, которые мы сохранили в самом последнем коммите.

Попробуем создать еще один файл, сделать коммит и после этого перейдем в другую ветку. Посмотрим, что изменится.

При переходе на ветку *master*, наш ранее созданный файл удаляется.

### *Примечание:*

Мы можем создавать еще ветки внутри других веток, если нам к примеру, требуется разделить функционал на двух программистов.

Итак, после того, как мы все доделали в созданной ветки, нам надо перегрузить в нашу главную ветку все изменения из дополнительной ветки.\

Это можно сделать двумя способами:

1. В главной ветке создаётся отдельный коммит, со всеми изменениями из дополнительной ветки.

2. В главную ветку переносится все коммиты с изменениями из второстепенной ветки.

Что выбирать, уже вам решать.

Команда для первого варианта:

Для начала, мы должны переместиться на ветку, куда хотим загрузить изменения. (**`git checkout master`**).

После этого, выполняем команду:

**`git merge <Имя ветки из которой хотим взять изменения>`**

После этого, в ветке *master*, у нас появятся все изменения, как отдельный коммит.

Команда для второго варианта:

Процедура аналогична. Мы переключаемся на ту ветку, в которую мы хотим поместить изменения (**`git checkout master`**).

После этого, выполняем команду:

**`git rebase <Имя ветки из которой хотим взять изменения>`**

После того, как мы локально произвели слияние двух веток, мы можем загрузить главную ветку на удаленный репозиторий, командой:

```
git push origin <Имя ветки, которую хотим загрузить>
```