

**МИНОБРНАУКИ РОССИИ**

**федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технологический университет «СТАНКИН»**

**(ФГБОУ ВО «МГТУ «СТАНКИН»)**

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | Институт цифровых интеллектуальных систем | Кафедра  компьютерных систем управления | |

Дисциплина «Основы системного программного обеспечения»

**Отчет по лабораторной работе №\_1\_**

Работа с системами контроля версий на примере Git Hub

|  |  |  |
| --- | --- | --- |
| **Выполнил**  **студент гр. АДБ-21-12:** | \_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_  *(дата) (подпись)* | **Алиев А.Э.** |
| **Проверил**  **к.т.н. доцент** | \_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_  *(дата) (подпись)* | **Ковалев И.А.** |

**Москва 2023 г.**

Краткая история Git

Как и многие замечательные вещи, Git начинался с, в некотором роде, разрушения во имя созидания и жарких споров. Ядро Linux — действительно очень большой открытый проект. Бо́льшую часть существования ядра Linux

(1991-2002) изменения к нему распространялись в виде патчей и

заархивированных файлов. В 2002 году проект перешёл на проприетарную РСКВ BitKeeper.

В 2005 году отношения между сообществом разработчиков ядра Linux и компанией, разрабатывавшей BitKeeper, испортились, и право бесплатного пользования продуктом было отменено. Это подтолкнуло разработчиков Linux (и в частности Линуса Торвальдса, создателя Linux) разработать собственную систему, основываясь на опыте, полученном за время использования BitKeeper. Основные требования к новой системе были следующими:

* Скорость
* Простота дизайна
* Поддержка нелинейной разработки (тысячи параллельных веток)
* Полная распределённость
* Возможность эффективной работы с такими большими проектами, как ядро Linux (как по скорости, так и по размеру данных)

С момента рождения в 2005 году Git развивался и эволюционировал, становясь проще и удобнее в использовании, сохраняя при этом свои первоначальные качества. Он невероятно быстр, очень эффективен для больших проектов, а также обладает превосходной системой ветвления для нелинейной разработки.

Основы Git

Так что же такое Git в двух словах? Эту часть важно усвоить, поскольку если вы поймёте, что такое Git, и каковы принципы его работы, вам будет гораздо проще пользоваться им эффективно. Изучая Git, постарайтесь освободиться от всего, что вы знали о других СКВ или постоянно пользуетесь, таких как Subversion или Perforce. В Git'е совсем не такие понятия об информации и работе с ней как в других системах, хотя пользовательский интерфейс очень похож. Знание этих различий защитит вас от путаницы при использовании Git'а.

## Git следит за целостностью данных

Перед сохранением любого файла Git вычисляет контрольную сумму, и она становится индексом этого файла. Поэтому невозможно изменить содержимое файла или каталога так, чтобы Git не узнал об этом. Эта функциональность встроена в сам фундамент Git'а и является важной составляющей его философии. Если информация потеряется при передаче или повредится на диске, Git всегда это выявит.

Механизм, используемый Git'ом для вычисления контрольных сумм, называется SHA-1 хешем. Это строка из 40 шестнадцатеричных символов (0- 9 и a-f), вычисляемая в Git'е на основе содержимого файла или структуры каталога. SHA-1 хеш выглядит примерно так:

*B6DABFD326E06D8896CBE83B3BD52645CCE69C28*

Работая с Git'ом, вы будете встречать эти хеши повсюду, поскольку он их очень широко использует. Фактически, в своей базе данных Git сохраняет всё не по именам файлов, а по хешам их содержимого.

## Чаще всего данные в Git только добавляются

Практически все действия, которые вы совершаете в Git'е, только добавляют данные в базу. Очень сложно заставить систему удалить данные или сделать что-то неотменяемое. Можно, как и в любой другой СКВ, потерять данные, которые вы ещё не сохранили, но как только они зафиксированы, их очень сложно потерять, особенно если вы регулярно отправляете изменения в другой репозиторий.

Поэтому пользоваться Git'ом — удовольствие, потому что можно экспериментировать, не боясь что-то серьёзно поломать

Область подготовленных файлов — это обычный файл, обычно хранящийся в каталоге Git'а, который содержит информацию о том, что должно войти в следующий коммит. Иногда его называют индексом (index), но в последнее время становится стандартом называть его областью подготовленных файлов (staging area).

Стандартный рабочий процесс с использованием Git'а выглядит примерно так:

1. Вы вносите изменения в файлы в своём рабочем каталоге.
2. Подготавливаете файлы, добавляя их слепки в область подготовленных файлов.
3. Делаете коммит, который берёт подготовленные файлы из индекса и помещает их в каталог Git'а на постоянное хранение.

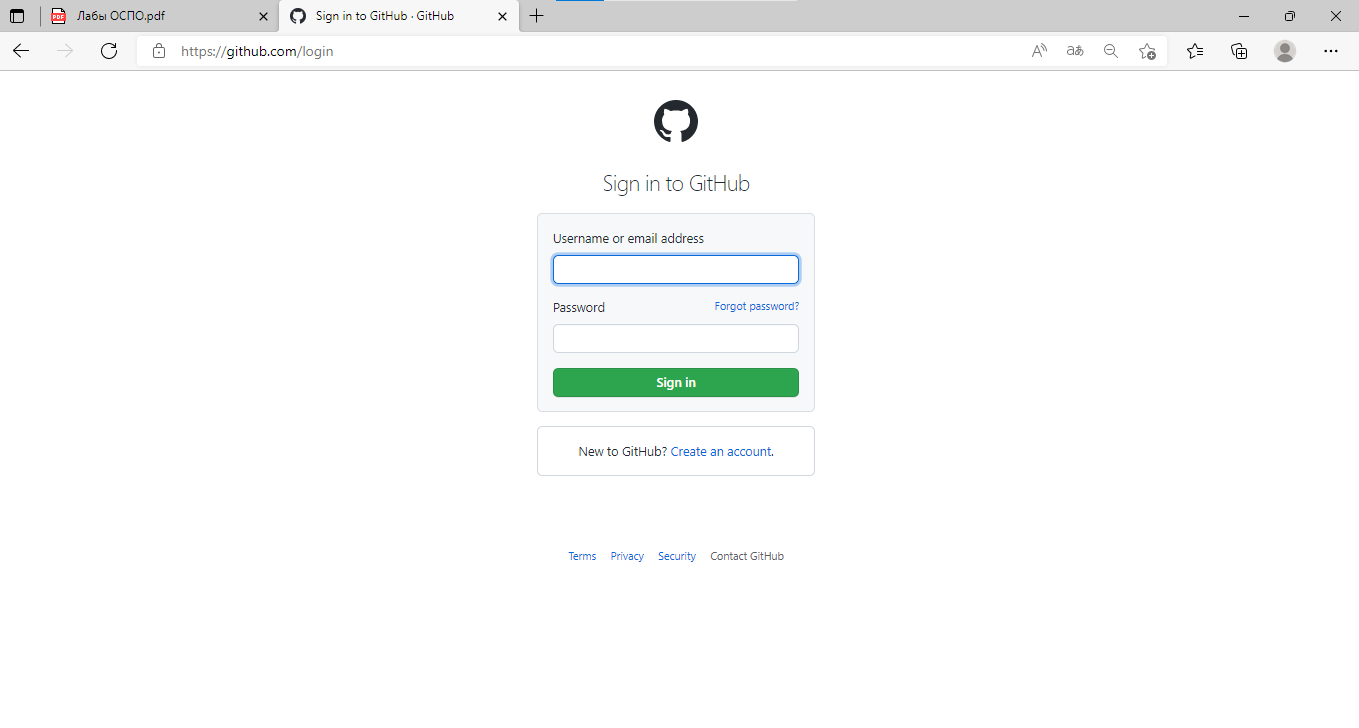
Если рабочая версия файла совпадает с версией в каталоге Git'а, файл считается зафиксированным. Если файл изменён, но добавлен в область подготовленных данных, он подготовлен. Если же файл изменился после выгрузки из БД, но не был подготовлен, то он считается изменённым.

**Индивидуальная задания**

## Создание учетной записи на github.com

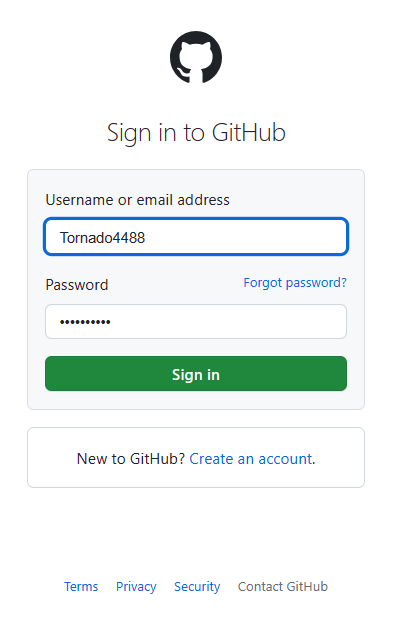
## 

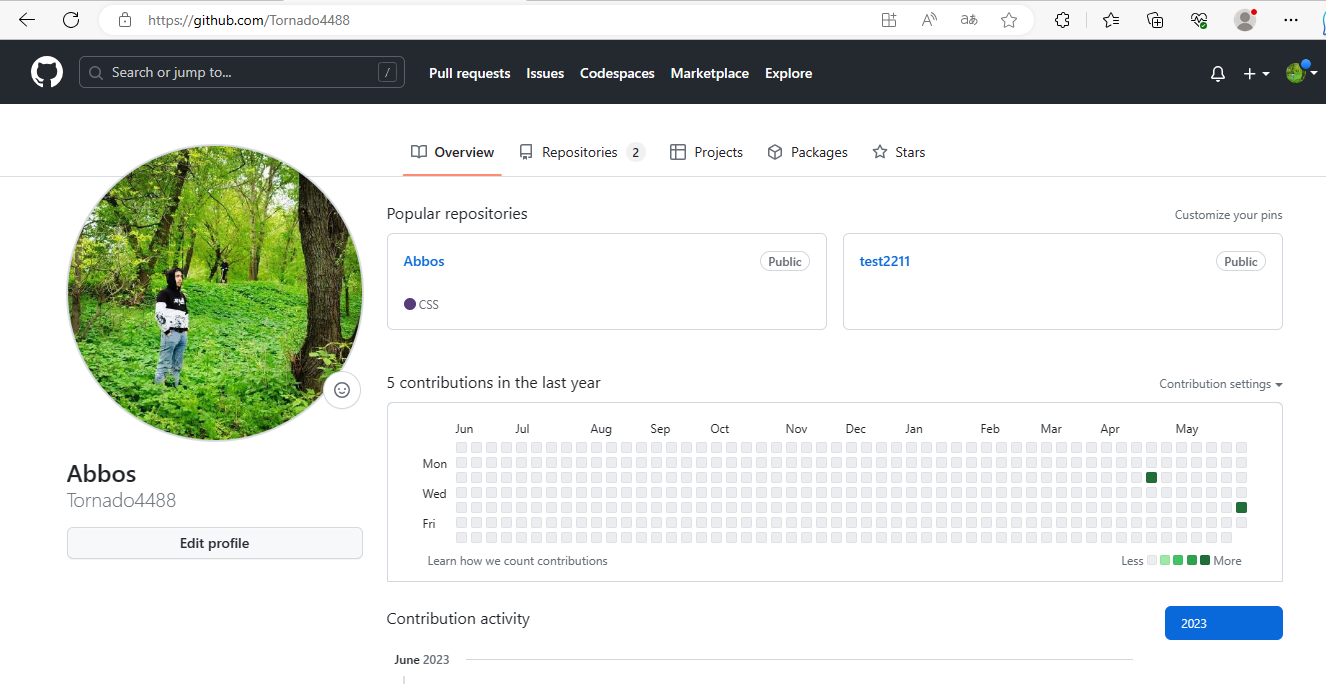
**Рисунок 1 Окно для регистрации в Git hub**



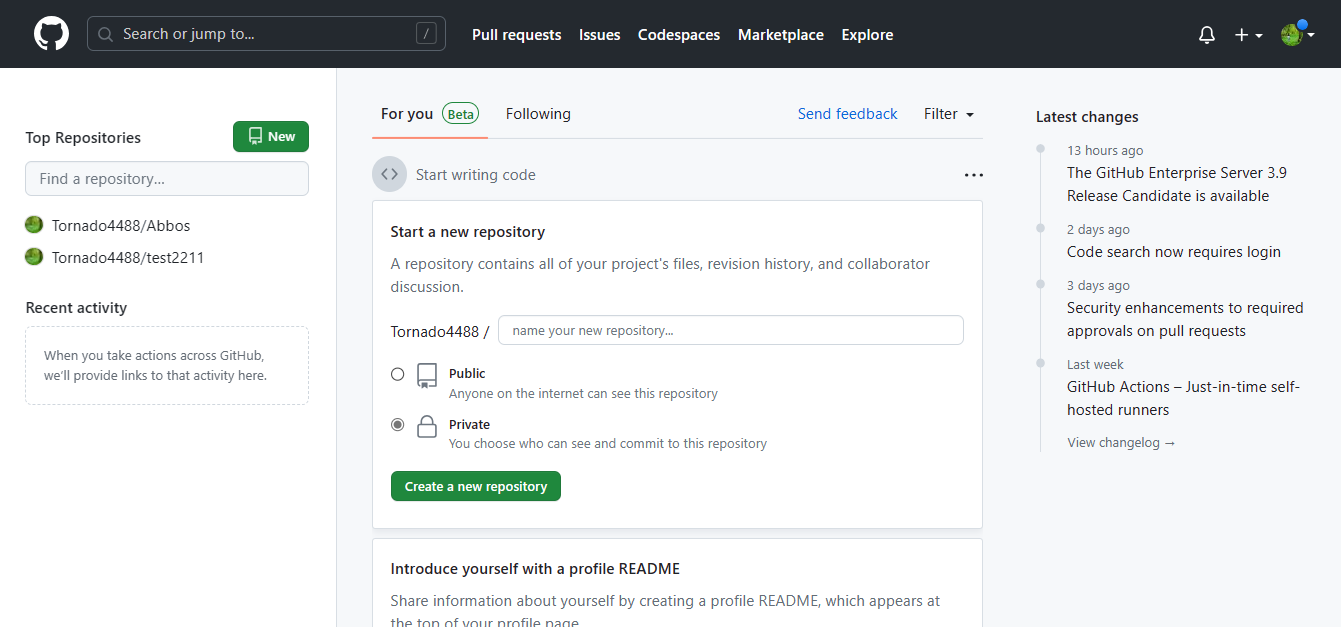
**Рисунок 2 Здесь должен написать логин парол чтобы зайти в Git hub**

Так как у нас есть уже созданный аккаунт и сразу напишем логин и пароль

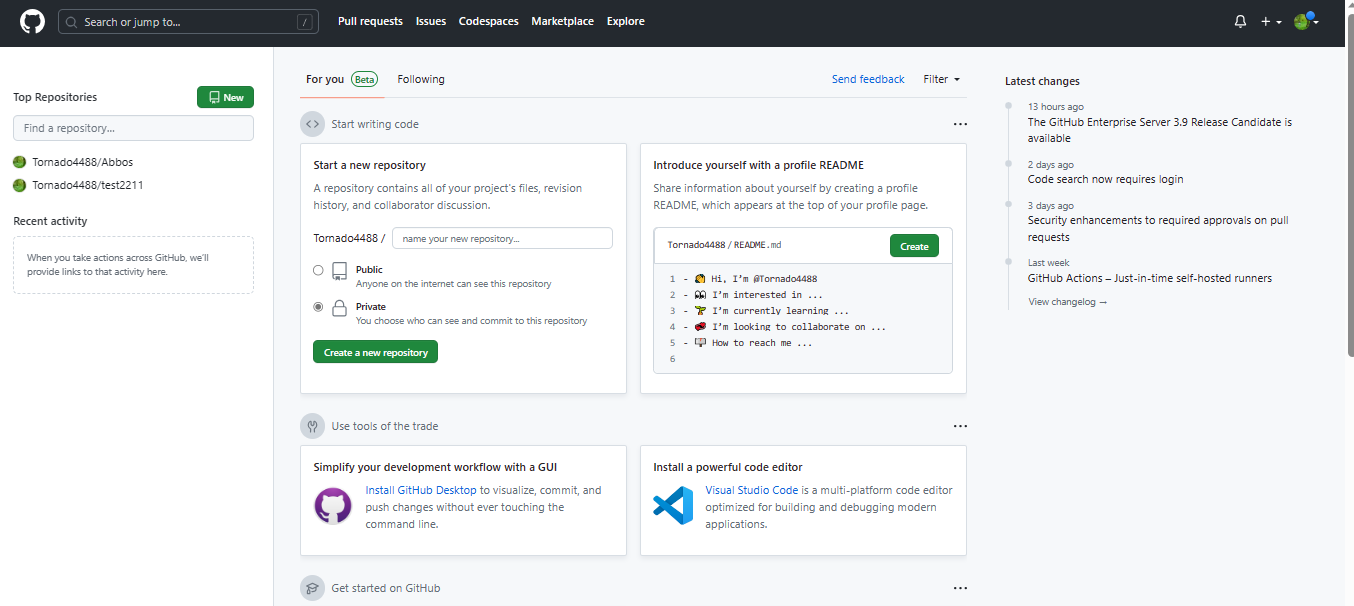




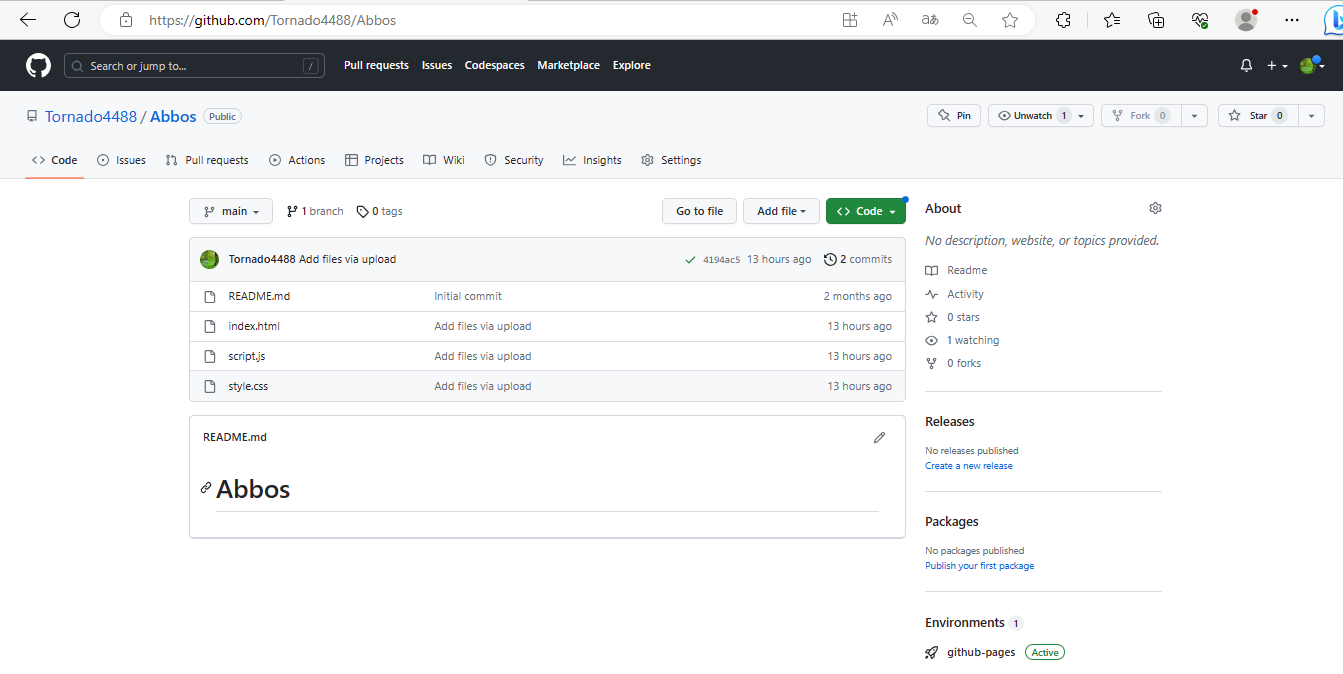
**Рисунок 3 Чтобы создать новый аккаунт нажимаем кнопку CREATE AN ACCOUNT**



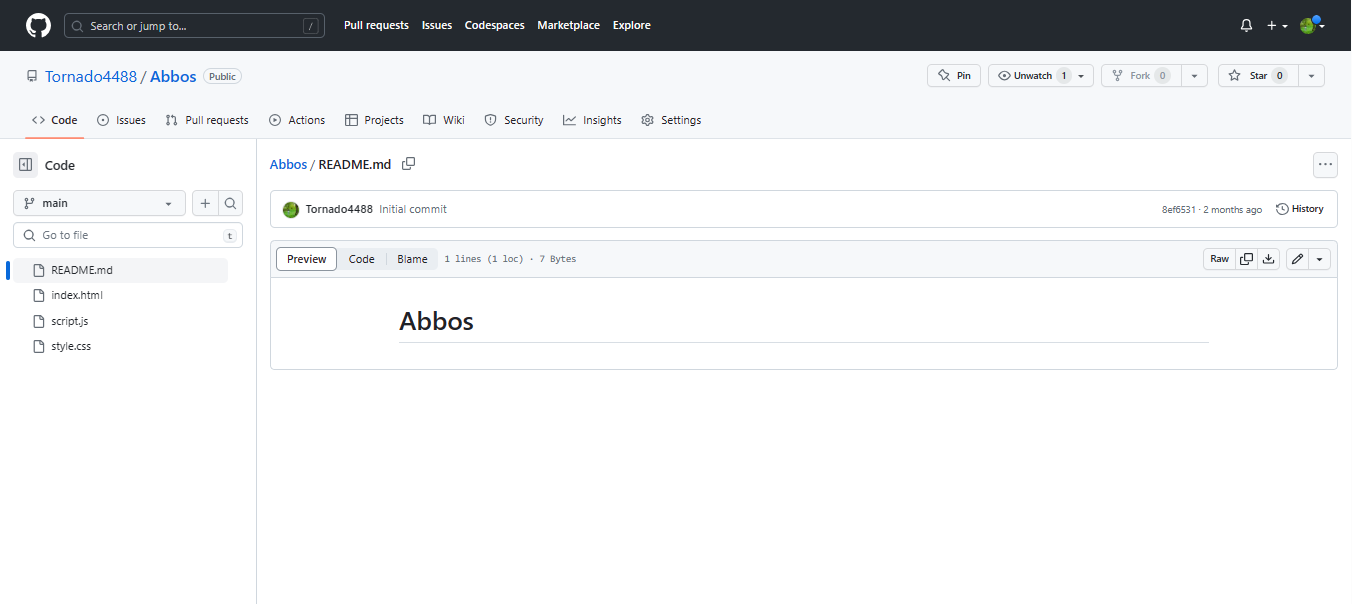
**Рисунок 4 Обший вид аккаунта Tornado4488**



**Рисунок 5 Заходим на эту репозиторию**



**Рисунок 6 Файлы который есть в репозитории Abbos**



**Рисунок 7 Файл README.md**

**Вывод:**

На этой лабораторной работе мы изучали как работать в Git hub.В ходе лабораторной работе мы создали аккаунт в Git hub <https://github.com/Tornado4488/Abbos/commits?author=Tornado4488> и в нём создали ветки , и на каждой ветки добавили тектовый файлы.