МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии
Департамент цифровых, робототехнических систем и электроники

Отчет по лабораторной работе № 1
по дисциплине «Программирование на Python»

Выполнил студент Ромащенко Данил группы ИВТ-б-о-24-1

Подпись студента_____
Работа защищена « »_____20__г.
Проверил Воронкин Р.А. _____
(подпись)

Ставрополь 2025

Тема: «Исследование основных возможностей Git и GitHub».

Цель работы: исследовать базовые возможности системы контроля версий Git и веб-сервиса для хостинга IT-проектов GitHub.

Ход работы

1) Была создан аккаунт на сайте GitHub.

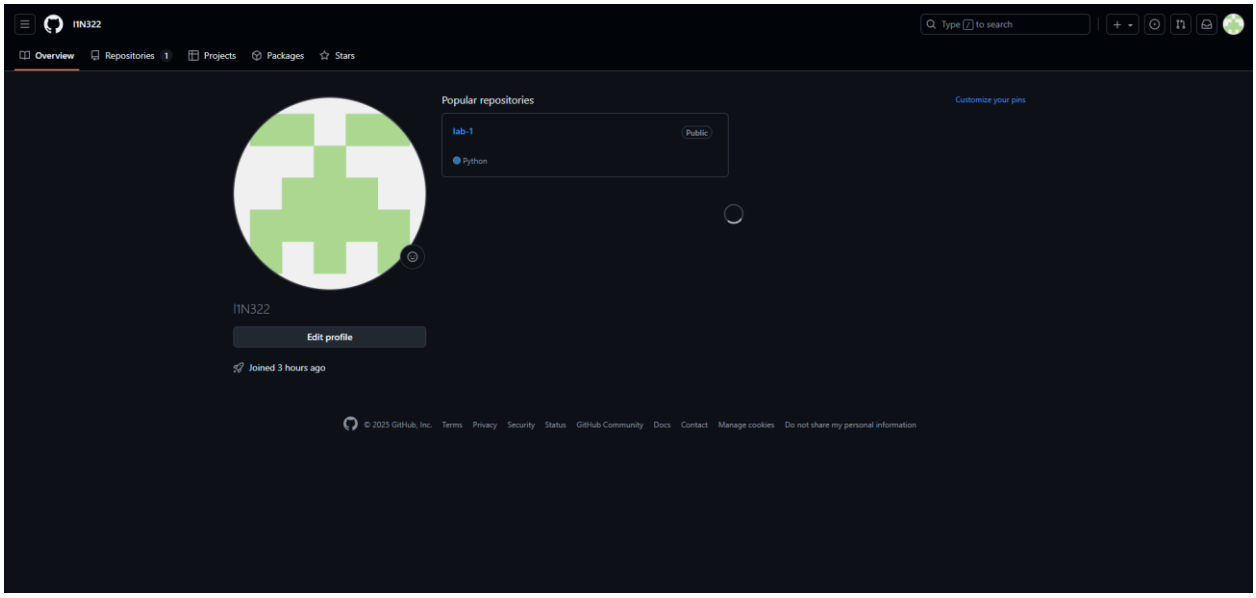

Рисунок 1. Профиль на сайте GitHub

2) Был создан репозиторий.

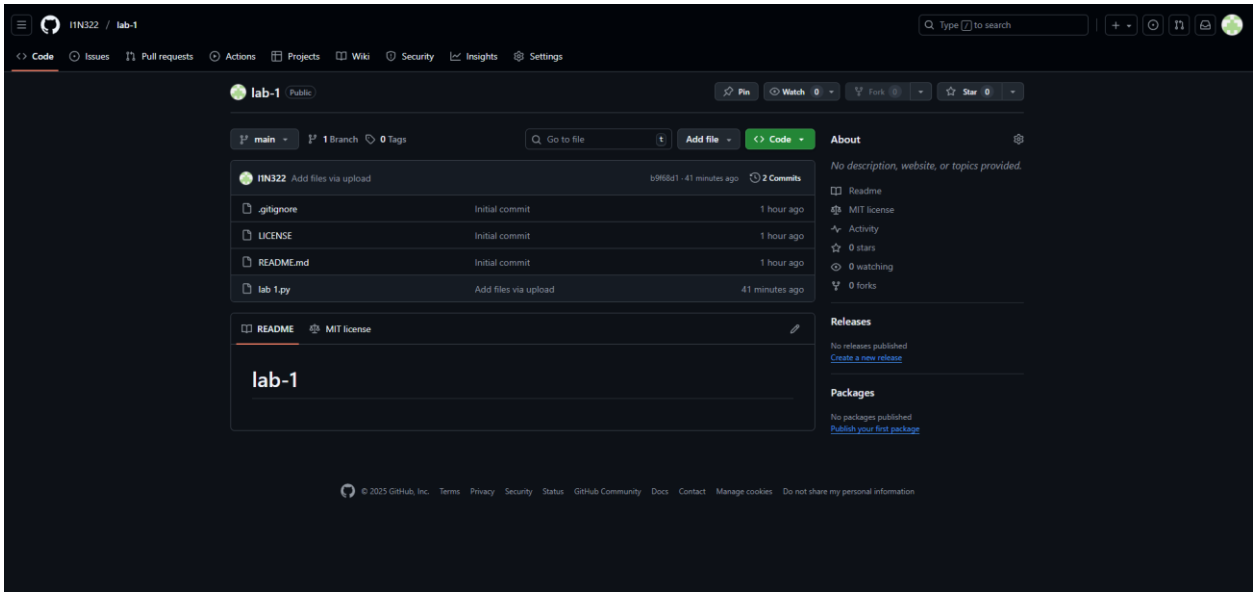

Рисунок 2. Созданный репозиторий

3) Было выполнено клонирование созданного репозитория на компьютер.

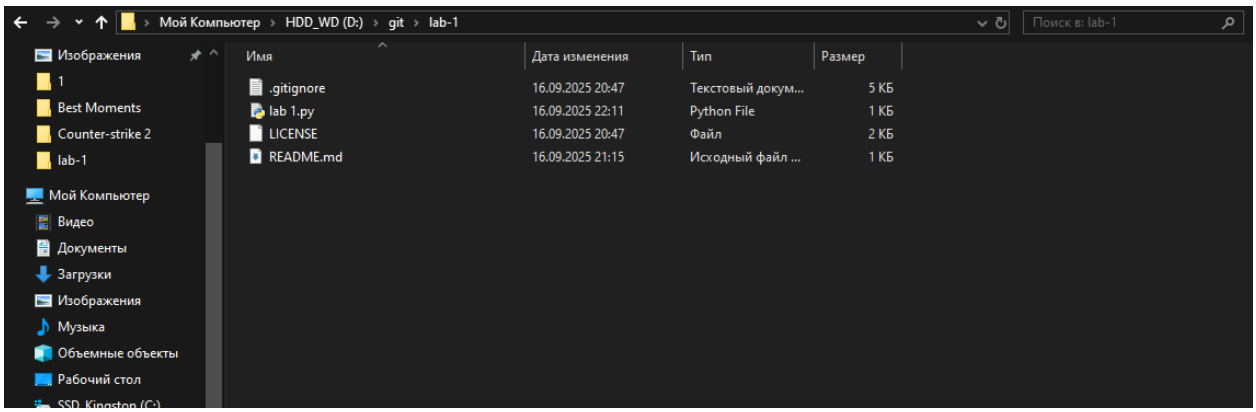

Рисунок 3. Клонирование репозитория

4) В файл README была добавлена информация о группе и ФИО студента.

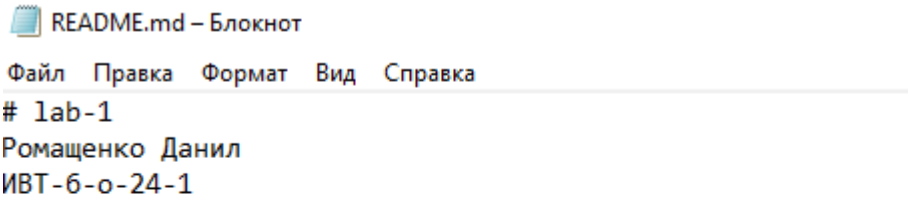

Рисунок 4. Добавленная информация о студенте

5) Была написана небольшая программа на языке программирования Python. В локальном репозитории были зафиксированы изменения при написании программы. Было сделано 7 коммитов.

```
USER@topPC MINGW64 /d/git/lab-1 (main)
$ git add "lab 1.py"

USER@topPC MINGW64 /d/git/lab-1 (main)
$ git commit -m "Первые изменения в программе"
[main 4b03f0a] Первые изменения в программе
 1 file changed, 1 insertion(+), 1 deletion(-)

USER@topPC MINGW64 /d/git/lab-1 (main)
$ ^C

USER@topPC MINGW64 /d/git/lab-1 (main)
$ git add "lab 1.py"

USER@topPC MINGW64 /d/git/lab-1 (main)
$ git commit -m "Вторые изменения"
[main ea82224] Вторые изменения
 1 file changed, 2 insertions(+)

USER@topPC MINGW64 /d/git/lab-1 (main)
$ git add "lab 1.py"
```

Рисунок 5. Фиксация изменений

6) Были зафиксированы изменения после добавления файла README.

```
USER@topPC MINGW64 /d/git/lab-1 (main)
$ git add README.md

USER@topPC MINGW64 /d/git/lab-1 (main)
$ git commit -m "Добавили файл README"
On branch main
Your branch is ahead of 'origin/main' by 9 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

Рисунок 6. Добавление файла README

7) Были отправлены изменения в исходный репозиторий.

```
USER@topPC MINGW64 /d/git/lab-1 (main)
$ git push
info: please complete authentication in your browser...
Enumerating objects: 31, done.
Counting objects: 100% (30/30), done.
Delta compression using up to 12 threads
Compressing objects: 100% (24/24), done.
Writing objects: 100% (26/26), 2.46 KiB | 840.00 KiB/s, done.
Total 26 (delta 12), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (12/12), done.
To https://github.com/l1N322/lab-1.git
   b9f68d1..c22d81b  main -> main
```

Рисунок 7. Отправление изменений в исходный репозиторий

Ответы на контрольные вопросы:

1) Что такое СКВ и каково ее назначение?

Система контроля версий (СКВ) – это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов.

2) В чем недостатки локальных и централизованных СКВ?

Многие люди в качестве метода контроля версий применяют копирование файлов в отдельную директорию. Такой подход очень распространён из-за его простоты, однако он невероятно сильно подвержен появлению ошибок. Можно легко забыть, в какой директории находитесь, и случайно изменить не тот файл или скопировать не те файлы, которые вы хотели. Это недостаток локальной СКВ.

Если сервер выйдет из строя на час, то в течение этого времени никто не сможет использовать контроль версий для сохранения изменений, над которыми работает, а также никто не сможет обмениваться этими изменениями с другими разработчиками. Если жёсткий диск, на котором хранится центральная БД, повреждён, а своевременные бэкапы отсутствуют, вы потеряете всё – всю историю проекта, не считая единичных снимков репозитория, которые сохранились на локальных машинах разработчиков. Это недостаток централизованной СКВ.

3) К какой СКВ относится Git?

Git относится к распределённым системам контроля версий.

4) В чем концептуальное отличие Git от других СКВ?

Основное отличие Git от любой другой СКВ – это подход к работе со своими данными. Концептуально, большинство других систем хранят информацию в виде списка изменений в файлах. Эти системы представляют хранимую информацию в виде набора файлов и изменений, сделанных в каждом файле, по времени. Git не хранит и не обрабатывает данные таким способом. Вместо этого, подход Git к хранению данных больше похож на набор снимков миниатюрной файловой системы. Каждый раз, когда вы делаете коммит, то есть сохраняете состояние своего проекта в Git, система запоминает, как выглядит каждый файл в этот момент, и сохраняет ссылку на

этот снимок. Для увеличения эффективности, если файлы не были изменены, Git не запоминает эти файлы вновь, а только создаёт ссылку на предыдущую версию идентичного файла, который уже сохранён. Git представляет свои данные как, скажем, поток снимков.

5) Как обеспечивается целостность хранимых данных в Git?

В Git для всего вычисляется хеш-сумма, и только потом происходит сохранение. В дальнейшем обращение к сохранённым объектам происходит по этой хеш-сумме. Это значит, что невозможно изменить содержимое файл или директории так, чтобы Git не узнал об этом. Данная функциональность встроена в Git на низком уровне и является неотъемлемой частью его основы. Вы не потеряете информацию во время её передачи и не получите повреждённый файл без ведома Git.

6) В каких состояниях могут находится файлы в Git? Как связаны эти состояния?

У Git есть три основных состояния, в которых могут находиться ваши файлы: зафиксированное (committed), изменённое (modified) и подготовленное (staged).

– Зафиксированный значит, что файл уже сохранён в вашей локальной базе.

– К изменённым относятся файлы, которые поменялись, но ещё не были зафиксированы.

– Подготовленные файлы – это изменённые файлы, отмеченные для включения в следующий коммит.

7) Что такое профиль пользователя в GitHub?

Профиль – это ваша публичная страница на GitHub, как и в социальных сетях.

8) Какие бывают репозитории в GitHub?

Репозитории бывают публичные и приватные. Публичные: любой желающий в Интернете может ознакомиться с этим хранилищем. Вы

выбираете, кто может взять на себя обязательства. Приватные: вы выбираете, кто может просматривать этот репозиторий и фиксировать его.

9) Укажите основные этапы модели работы с GitHub.

– Вы изменяете файлы в вашей рабочей директории.

– Вы выборочно добавляете в индекс только те изменения, которые должны попасть в следующий коммит, добавляя тем самым снимки только этих изменений в область подготовленных файлов.

– Когда вы делаете коммит, используются файлы из индекса как есть, и этот снимок сохраняется в вашу Git-дирктурию.

10) Как осуществляется первоначальная настройка Git после установки?

Чтобы убедиться, что Git был успешно установлен используем команду git version

Если она сработала, давайте добавим в настройки Git ваше имя, фамилию и адрес электронной почты, связанный с вашей учетной записью GitHub:

git config --global user.name <YOUR_NAME>

git config --global user.email <EMAIL>

11) Опишите этапы создания репозитория в GitHub.

В правом верхнем углу, рядом с аватаром есть кнопка с плюсиком, нажимая которую мы переходим к созданию нового репозитория.

В результате будет выполнен переход на страницу создания репозитория. Наиболее важными на ней являются следующие поля:

Имя репозитория. Оно может быть любое, необязательно уникальное во всем github, потому что привязано к вашему аккаунту, но уникальное в рамках тех репозиториев, которые вы создавали. Описание (Description). Можно оставить пустым. Public/private. Выбираем открытый (Public), НЕ ставим галочку “Initialize this repository with a README” (В README потом будет лежать какая-то основная информация, что же такое ваш проект и как с ним работать). .gitignore и LICENSE можно сейчас не выбирать. После заполнения этих полей нажимаем кнопку Create repository.

12) Какие типы лицензий поддерживаются GitHub при создании репозитория?

Apache License 2.0, GNU General Public License v3.0, MIT License, BSD 2-Clause "Simplified" License, BSD 3-Clause "New" or "Revised" License, Boost Software License 1.0, Creative Commons Zero v1.0 Universal, Eclipse Public License 2.0, GNU Affero General Public License v3.0, GNU General Public License v2.0, GNU Lesser General Public License v2.1, Mozilla Public Licens e2. Unlicense.

13) Как осуществляется клонирование репозитория GitHub? Зачем нужно клонировать репозиторий?

После создания репозитория его необходимо клонировать на ваш компьютер. Для этого на странице репозитория необходимо найти кнопку Clone или Code и щелкнуть по ней, чтобы отобразить адрес репозитория для клонирования.

Откройте командную строку или терминал и перейдите в каталог, куда вы хотите скопировать хранилище. Затем напишите git clone и введите адрес.

Стандартный подход к работе с проектом состоит в том, чтобы иметь локальную копию репозитория и фиксировать ваши изменения в этой копии, а не в удаленном репозитории, размещенном на GitHub. Этот локальный репозиторий имеет полную историю версий проекта, которая может быть полезна при разработке без подключения к интернету. После того, как вы что-то изменили в локальном, вы можете отправить свои изменения в удаленный репозиторий, чтобы сделать их видимыми для других разработчиков.

14) Как проверить состояние локального репозитория Git?

Чтобы проверить состояние вашего репозитория введите команду git status.

15) Как изменяется состояние локального репозитория Git после выполнения следующих операций: добавления/изменения файла в локальный репозиторий Git; добавления нового/измененного файла под версионный контроль с помощью команды git add; фиксации (коммита) изменений с

помощью команды git commit и отправки изменений на сервер с помощью команды git push?

Когда вы добавляете новый файл или вносите изменения в существующий файл, Git отслеживает эти изменения, но они ещё не зафиксированы.

Изменения, внесённые в файл, добавляются в индекс с помощью команды git add — специальную промежуточную область между рабочей директорией и репозиторием. После добавления в индекс файлы получают статус подготовленных.

Изменения, добавленные в индекс, сохраняются в репозитории как отдельный коммит (git commit). Таким образом, Git ведёт историю изменений проекта. Коммит снабжается сообщением (комментарием), в котором указывается, что за изменения были внесены.

Изменения, сделанные локально, отправляются на удалённый репозиторий с помощью команды git push. Это позволяет сохранить историю изменений, которые были сделаны локально, в удалённом репозитории.

16) У вас имеется репозиторий на GitHub и два рабочих компьютера,с помощью которых Вы можете осуществлять работу над некоторым проектомс использованием этого репозитория. Опишите последовательность команд, с помощью которых оба локальных репозитория, связанных с репозиторием GitHub будут находиться в синхронизированном состоянии.

Примечание: описание необходимо начать с команды git clone.

– Клонируем репозиторий на каждый из покмпьютеров с помошью команды git clone и ссылки на репозиторий.

– Создаем и коммитим изменения с помощью команд git add и git commit.

– Отправляем изменения на GitHub с помощью команды git push.

– Синхронизируем изменения, созданные другим рабочим компьютером с помощью команды git pull.

17) GitHub является не единственным сервисом, работающим с Git.

Какие сервисы еще Вам известны? Приведите сравнительный анализ одного из таких сервисов с GitHub.

Кроме GitHub, с системой контроля версий Git работают, например, такие сервисы: GitLab и Bitbucket.

Bitbucket также предназначен для хранения git-репозиториев и управления ими. Отличие сервиса от GitHub в том, что он «заточен» на использование приватных репозиториев.

18) Интерфейс командной строки является не единственным и далеко не самым удобным способом работы с Git. Какие Вам известны программные средства с графическим интерфейсом пользователя для работы с Git? Приведите как реализуются описанные в лабораторной работе операции Git с помощью одного из таких программных средств.

Для работы с Git существуют следующие программные средства с графическим интерфейсом: GitHub Desktop, SourceTree, TortoiseGit, GitKraken, Git GUI.

Процесс создания нового коммита через Git GUI:

– Запустить интерфейс и открыть существующий репозиторий или создать новый.

– В главном окне перейти в раздел Unstaged Changes.

– Выбрать файлы, которые необходимо включить в коммит, с помощью кнопки Stage Changed.

– В поле сообщения коммита ввести описание изменений согласно принятому формату.

– Подтвердить создание коммита кнопкой Commit.

Вывод: в ходе работы были исследованы базовые возможности системы контроля версий Git и веб-сервиса для хостинга IT-проектов GitHub.