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
дисциплины
«Программирование на python»

Выполнил:
Червиченко Иван Денисович
1 курс, группа ИВТ-б-о-24-2,
09.03.01 «Информатика и вычислительная техника»,
направленность (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения

_____
(подпись)

Проверила:
доцент департамента цифровых, робототехнических систем и электроники института перспективной инженерии Воронкин Р.А

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2025 г.

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

Ход работы:

1. Регистрация на GitHub

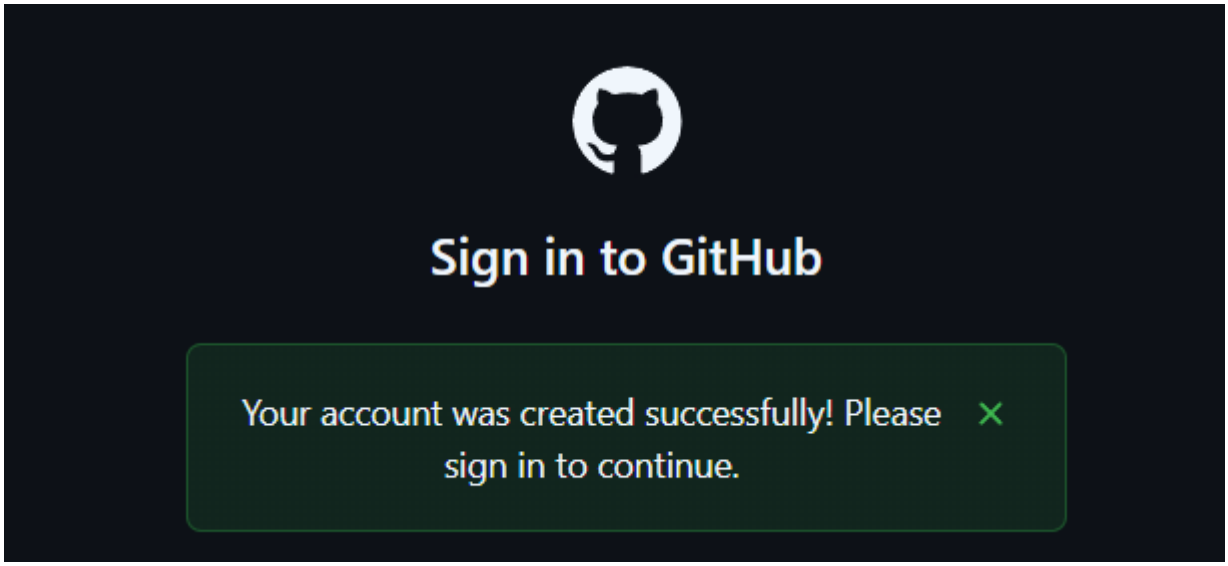

Рисунок 1. Успешная регистрация на GitHub

2. Создание репозитория

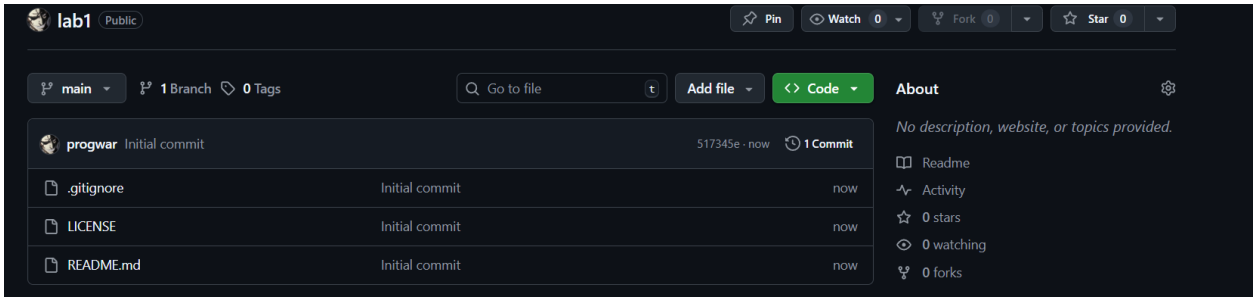

Рисунок 2. Созданный репозиторий

3. Клонирование репозитория на компьютер

```
PS C:\Windows\system32> cd "C:\Program Files\github"
PS C:\Program Files\github> git clone https://github.com/progwar/lab1.git
Cloning into 'lab1'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (5/5), done.
PS C:\Program Files\github>
```

Рисунок 3. Клонирование репозитория

4. Был изменён файл README.md

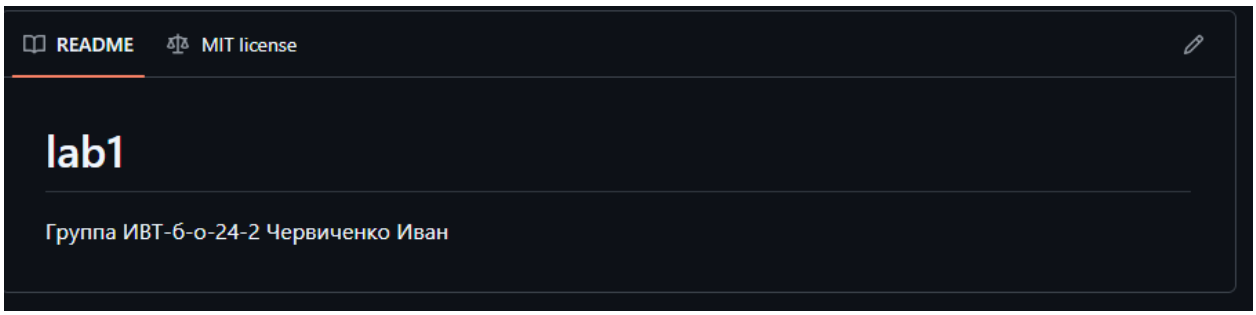

Рисунок 4. Изменённый файл на GitHub

5. Была написана программа на Python

```
print("x,y,z,w")
for x in range(0, 2):
    for y in range(0, 2):
        for z in range(0, 2):
            for w in range(0, 2):
                if not((x == (w or y)) or ((w <= z) and (y <= w))):
                    print(x, y, z, w)
```

Рисунок 5. Написанная программа

6. По мере написания программы было сделано 7 коммитов

```
PS D:\github\lab1> git add pylab1.py
PS D:\github\lab1> git commit -m "2 commit"
[main 05f7e3f] 2 commit
 1 file changed, 1 insertion(+)
PS D:\github\lab1> git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 296 bytes | 296.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/progwar/lab1.git
   cd94af2..05f7e3f  main -> main
PS D:\github\lab1>
```

```
PS D:\github\lab1> git add pylab1.py
PS D:\github\lab1> git commit -m "3 commit"
[main ac4c0bf] 3 commit
 1 file changed, 1 insertion(+)
PS D:\github\lab1> git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 303 bytes | 303.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/progwar/lab1.git
   05f7e3f..ac4c0bf  main -> main
PS D:\github\lab1>
```

```
PS D:\github\lab1> git add pylab.py
fatal: pathspec 'pylab.py' did not match any files
PS D:\github\lab1> git add pylab1.py
PS D:\github\lab1> git commit -m "4 commit"
[main ef59f18] 4 commit
 1 file changed, 1 insertion(+)
PS D:\github\lab1> git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 308 bytes | 308.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/progwar/lab1.git
   ac4c0bf..ef59f18  main -> main
PS D:\github\lab1>
```

```
PS D:\github\lab1> git add pylab1.py
PS D:\github\lab1> git commit -m "5 commit"
[main d8621c1] 5 commit
 1 file changed, 1 insertion(+)
PS D:\github\lab1> git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 305 bytes | 305.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/progwar/lab1.git
   ef59f18..d8621c1  main -> main
PS D:\github\lab1>
```

```
PS D:\github\lab1> git add pylab1.py
PS D:\github\lab1> git commit -m "6 commit"
[main c148076] 6 commit
 1 file changed, 1 insertion(+)
PS D:\github\lab1> git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 332 bytes | 332.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/progwar/lab1.git
   d8621c1..c148076  main -> main
PS D:\github\lab1>
```

```
PS D:\github\lab1> git add pylab1.py
PS D:\github\lab1> git commit -m "7 commit"
[main 42d46d5] 7 commit
 1 file changed, 1 insertion(+)
PS D:\github\lab1> git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 302 bytes | 302.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/progwar/lab1.git
   c148076..42d46d5  main -> main
PS D:\github\lab1>
```

Рисунок 6. Коммит программы

Рисунок 7. Добавленные изменения на GitHub

Контрольные вопросы:

1) Что такое СКВ и каково её 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.

СКВ даёт возможность возвращать отдельные файлы к прежнему виду, возвращать к прежнему состоянию весь проект, просматривать происходящие со временем изменения, определять, кто последним вносил изменения во внезапно переставший работать модуль, кто и когда внёс в код какую-то ошибку, и многое другое.

2) В чём недостатки локальных и централизованных СКВ?

Локальная СКВ сильно подвержена появлению ошибок. Можно легко забыть, в какой директории вы находитесь, и случайно изменить не тот файл или скопировать не те файлы, которые вы хотели.

Централизованная СКВ Самый очевидный минус - это единая точка отказа, представленная централизованным сервером. Если этот 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.

3) К какой СКВ относится Git?

Git относится к РСКВ(Распределённая система контроля версий)

4) В чём концептуальное отличие Git от других СКВ?

Основное отличие Git от любой другой СКВ (включая Subversion и её собратьев) - это подход к работе со своими данными. Концептуально, большинство других систем хранят информацию в виде списка изменений в файлах.

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. Git представляет свои данные как 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 файла или директории так, чтобы Git не узнал об этом.

Механизм, которым пользуется Git при вычислении хеш-сумм, называется SHA-1 хеш. Это строка длинной в 40 шестнадцатеричных символов (0-9 и a-f), она вычисляется на основе содержимого файла или структуры каталога. SHA-1 хеш выглядит примерно так:

24b9da6552252987aa493b52f8696cd6d3b00373

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.

Есть три основных секции проекта Git: Git-директория (Git directory), рабочая директория (working directory) и область подготовленных файлов (staging area)

7) Что такое профиль пользователя в GitHub?

Профиль - это ваша публичная страница на GitHub, как и в социальных сетях. Когда вы ищете работу в качестве программиста, работодатели могут посмотреть ваш профиль GitHub и принять его во внимание, когда будут решать, брать вас на работу или нет.

8) Какие бывают репозитории в GitHub?

- Публичные (Public): Видны всем пользователям интернета. Любой может просмотреть код и клонировать репозиторий.
- Приватные (Private): Видны только владельцу и explicitly добавленным collaborators.

9) Укажите основные этапы модели работы с GitHub.

Рассмотрим область GitHub. В нем есть два хранилища:

• upstream - это оригинальный репозиторий проекта, который вы скопировали,

• origin - ваш fork (копия) на GitHub, к которому у вас есть полный доступ.

Чтобы перенести изменения с вашей копии в исходному репозиторий проекта, вам нужно сделать запрос на извлечение.

Если вы хотите внести небольшие изменения в свою копию (fork), вы можете использовать веб-интерфейс GitHub. Однако такой подход не удобен при разработке программ, поскольку вам часто приходится запускать и отлаживать их локально. Стандартный способ создать локальный удаленного репозитория и работать с ним локально, периодически внося изменения в удаленный репозиторий.

10) Как осуществляется первоначальная настройка Git после установки?

Чтобы убедиться, что Git был успешно установлен, введите команду ниже в терминале, чтобы отобразить текущую версию вашего Git:

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 GitHub.

В правом верхнем углу, рядом с аватаром есть кнопка с плюсом, 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 в GitHub при создании репозитория?

GitHub поддерживает огромное количество лицензий с помощью шаблонов. Наиболее популярные:

- MIT License: Очень разрешительная лицензия.
- Apache License 2.0
- GNU General Public License v3.0 (GPL-3.0): Копилефт-лицензия.
- BSD 2-Clause "Simplified" License
- BSD 3-Clause "New" or "Revised" License
- The Unlicense: Публичное достояние.

И многие другие.

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:

git clone <адрес репозитория>

Теперь у вас есть локальное хранилище проекта. Клонирование создает полную локальную копию удаленного репозитория со всей его историей. Это необходимо, чтобы начать работу над проектом на своем компьютере.

14) Как проверить состояние локального репозитория Git?

Перейдите в каталог хранилища и посмотрите на содержимое. Локальный репозиторий включает в себя все те же файлы, ветки и историю коммитов, как и удаленный репозиторий. Введите эту команду, чтобы проверить состояние вашего репозитория:

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?

- Добавления/изменения файла: Файл переходит в состояние Modified (если он уже отслеживался) или Untracked (если это новый файл). git status покажет эти изменения.
- git add: Файлы переходят из состояния Modified/Untracked в состояние Staged. Их изменения помещаются в "staging area".
- git commit: Все проиндексированные (staged) изменения фиксируются, создается новый коммит (снимок состояния). Файлы переходят в состояние Committed. Staging area очищается.
- git push: Локальные коммиты, которых еще нет на удаленном сервере, отправляются на него. Состояния удаленного и локального репозиториев синхронизируются.

16) У Вас имеется репозиторий на GitHub и два рабочих компьютера, с помощью которых Вы можете осуществлять работу над некоторым проектом с 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.

На первом компьютере:

1. git clone <url_репозитория_на_GitHub> (создает локальную копию)
2. (работа над файлами, затем...) git add .
3. git commit -m "Мой коммит"
4. git push (отправляет изменения на GitHub)

На втором компьютере:

1. git clone <url_репозитория_на_GitHub> (если еще не клонирован)

2. (Перед началом работы) git pull (получает все изменения, сделанные на Компьютере 1 и отправленные на GitHub)

3. (работа над файлами, затем...) git add .

4. git commit -m "Мой коммит с компьютера 2"

5. git push

17) GitHub является не единственным сервисом, работающим с Git. Какие сервисы еще Вам известны? Проведите сравнительный анализ одного из таких сервисов с GitHub.

Известные альтернативы:

- GitLab
- Bitbucket
- Azure DevOps Server

Сравнительный анализ GitLab vs GitHub:

Интерфейс:

1. (Hub) Чистый, минималистичный, интуитивный.

2. (Lab) Функциональный, но может казаться перегруженным.

Философия:

1. (Hub) Социальная платформа для open-source. Де-факто стандарт.

2. (Lab) Единая DevOps-платформа "всё в одном".

Фокус:

1. (Hub) Публичные проекты и open-source.

2. (Lab) Корпоративный сектор и приватные проекты.

18) Какие Вам известны программные средства с графическим интерфейсом пользователя для работы с Git?

Популярные GUI-клиенты:

- GitHub Desktop
- GitKraken
- Sourcetree
- TortoiseGit (интегрируется в проводник Windows)

- Встроенные в IDE: JetBrains IDE (IntelliJ IDEA, PyCharm), Visual Studio Code.

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