

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития  
Кафедра инфокоммуникаций

**ОТЧЕТ**  
**ПО ПРАКТИЧЕСКОЙ РАБОТЕ №1**  
**дисциплины «Программирование на Python»**

Выполнил:  
Данилецкий Дмитрий Витальевич  
2 курс, группа ИВТ-б-о-22-1,  
09.03.01 «Информатика и  
вычислительная техника»,  
направленность (профиль)  
«Программное обеспечение средств  
вычислительной техники и  
автоматизированных систем», очная  
форма обучения

---

(подпись)

Руководитель практики:  
Воронкин Р. А., канд. технических  
наук, доцент кафедры  
инфокоммуникаций

---

(подпись)

Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_

Ставрополь, 2023 г.

Тема: Исследование основных возможностей Git и GitHub.

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

### Краткие теоритические сведения

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.

Локальные СКВ (системы контроля версий) - это инструменты, используемые разработчиками для отслеживания и управления изменениями в их программном коде на локальном компьютере.

Централизованные СКВ (системы контроля версий) - это тип СКВ, где весь исходный код и его история хранятся в одном центральном репозитории, и разработчики работают с этим репозиторием для совместной работы и управления версиями кода. Примером централизованных СКВ является Subversion (SVN).

Распределённые СКВ (системы контроля версий) - это тип СКВ, в котором каждый разработчик имеет свою собственную копию полного репозитория, и они могут работать над кодом независимо, а затем обмениваться изменениями между собой. Примером распределённых СКВ является Git.

Git представляет свои данные как, скажем, поток снимков.

В Git для всего вычисляется хеш-сумма, и только потом происходит сохранение. Механизм, которым пользуется Git при вычислении хеш-сумм, называется SHA-1 хеш. Это строка длиной в 40 шестнадцатеричных символов (0-9 и a-f), она вычисляется на основе содержимого файла или структуры каталога.

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.

Git-директория — это то место, где Git хранит метаданные и базу объектов вашего проекта. Это самая важная часть Git, и это та часть, которая копируется при клонировании репозитория с другого компьютера.

Рабочая директория является снимком версии проекта. Файлы распаковываются из сжатой базы данных в Git-директории и располагаются на диске, для того чтобы их можно было изменять и использовать.

Область подготовленных файлов — это файл, обычно располагающийся в вашей Git-директории, в нём содержится информация о том, какие изменения попадут в следующий коммит. Эту область ещё называют “индекс”, однако называть её stage-область также общепринято.

## Ход работы

### 1. Создал новый общедоступный репозиторий с MIT лицензией.

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 \* Daniletskiy-D / Repository name \* LAB-1

LAB-1 is available.

Great repository names are short and memorable. Need inspiration? How about [ideal-system](#) ?

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: C++

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: MIT License

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

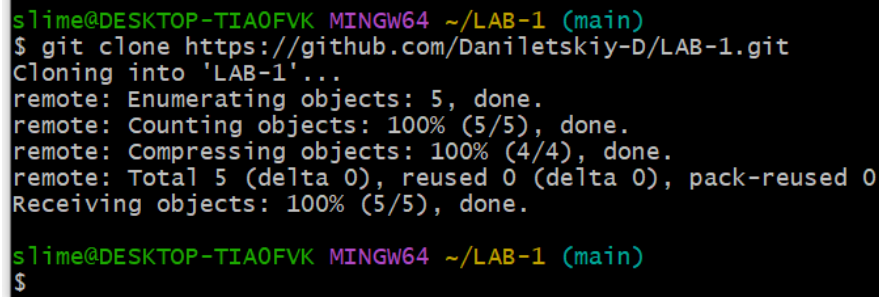
This will set [main](#) as the default branch. Change the default name in your [settings](#).

① You are creating a public repository in your personal account.

[Create repository](#)

Рисунок 1. Новый репозиторий

2. Выполнил клонирование созданного репозитория на рабочий компьютер.

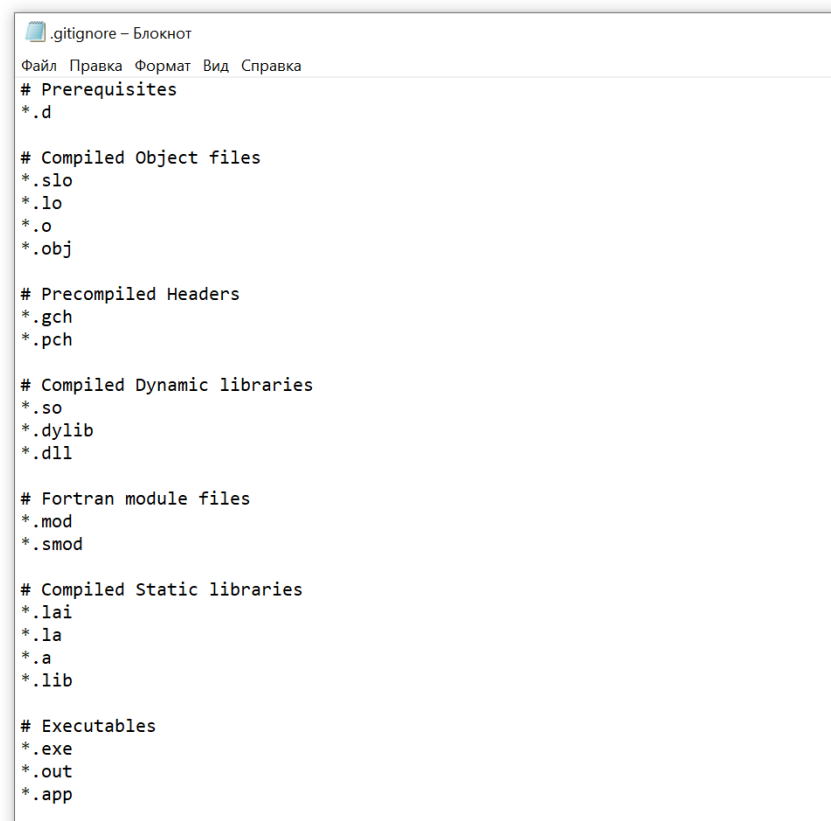


```
slime@DESKTOP-TIA0FVK MINGW64 ~/LAB-1 (main)
$ git clone https://github.com/Daniletskiy-D/LAB-1.git
Cloning into 'LAB-1'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.

slime@DESKTOP-TIA0FVK MINGW64 ~/LAB-1 (main)
$
```

Рисунок 2. Копирование репозитория

3. Дополнил файл .gitignore необходимыми правилами для выбранного языка программирования и интегрированной среды разработки.



```
.gitignore - Блокнот
Файл Правка Формат Вид Справка
# Prerequisites
*.d

# Compiled Object files
*.slo
*.lo
*.o
*.obj

# Precompiled Headers
*.gch
*.pch

# Compiled Dynamic libraries
*.so
*.dylib
*.dll

# Fortran module files
*.mod
*.smod

# Compiled Static libraries
*.lai
*.la
*.a
*.lib

# Executables
*.exe
*.out
*.app
```

Рисунок 3. Дополненный файл

4. Добавил в файл README.md информацию о группе и ФИО студента, выполняющего лабораторную работу.

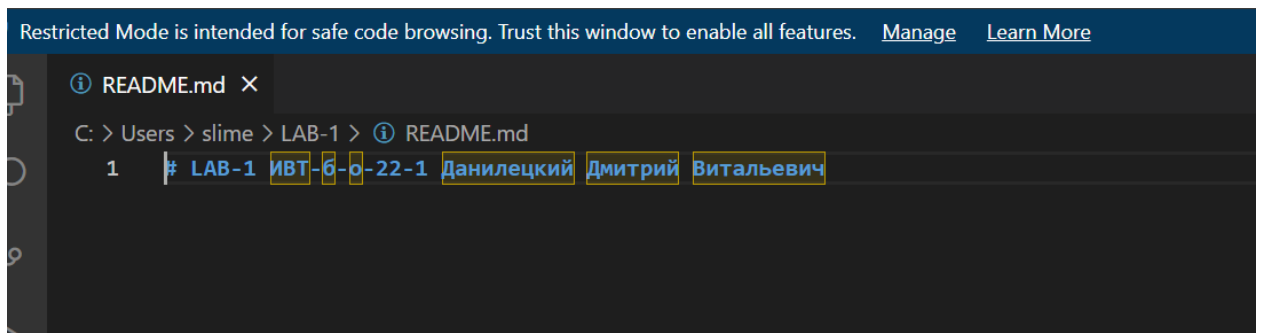


Рисунок 4. Изменённый README

5. Написал небольшую программу на C++. Фиксировал изменения при написании программы в локальном репозитории.

```
slime@DESKTOP-TIA0FVK MINGW64 ~/LAB-1 (main)
$ git commit -m "Изменение 2"
[main 5b6e869] Изменение 2
1 file changed, 10 insertions(+), 1 deletion(-)

slime@DESKTOP-TIA0FVK MINGW64 ~/LAB-1 (main)
$ git add .

slime@DESKTOP-TIA0FVK MINGW64 ~/LAB-1 (main)
$ git commit -m "Изменение 3"
[main fc31f51] Изменение 3
1 file changed, 7 insertions(+)

slime@DESKTOP-TIA0FVK MINGW64 ~/LAB-1 (main)
$ git add .

slime@DESKTOP-TIA0FVK MINGW64 ~/LAB-1 (main)
$ git commit -m "Изменение 4"
[main c748b8f] Изменение 4
1 file changed, 3 insertions(+)

slime@DESKTOP-TIA0FVK MINGW64 ~/LAB-1 (main)
$ git add .

slime@DESKTOP-TIA0FVK MINGW64 ~/LAB-1 (main)
$ git commit -m "Изменение 5"
[main 1880281] Изменение 5
1 file changed, 3 insertions(+)

slime@DESKTOP-TIA0FVK MINGW64 ~/LAB-1 (main)
$ git add .

slime@DESKTOP-TIA0FVK MINGW64 ~/LAB-1 (main)
$ git commit -m "Изменение 6"
[main a9752f5] Изменение 6
1 file changed, 1 insertion(+)

slime@DESKTOP-TIA0FVK MINGW64 ~/LAB-1 (main)
$ git add .

slime@DESKTOP-TIA0FVK MINGW64 ~/LAB-1 (main)
$ git commit -m "Изменение 7"
[main 28bac16] Изменение 7
1 file changed, 2 insertions(+), 1 deletion(-)

slime@DESKTOP-TIA0FVK MINGW64 ~/LAB-1 (main)
$
```

Рисунок 5. Фиксирование изменений

```
nothing to commit, working tree clean

s\lme@DESKTOP-TIA0FVK MINGW64 ~/LAB-1 (main)
$ git push
Enumerating objects: 24, done.
Counting objects: 100% (24/24), done.
Delta compression using up to 12 threads
Compressing objects: 100% (21/21), done.
Writing objects: 100% (22/22), 2.17 KiB | 2.17 MiB/s, done.
Total 22 (delta 11), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (11/11), done.
To https://github.com/Daniletskiy-D/LAB-1.git
594080f..28bac16  main -> main

s\lme@DESKTOP-TIA0FVK MINGW64 ~/LAB-1 (main)
$ |
```

Рисунок 6. Отправка файлов на Github

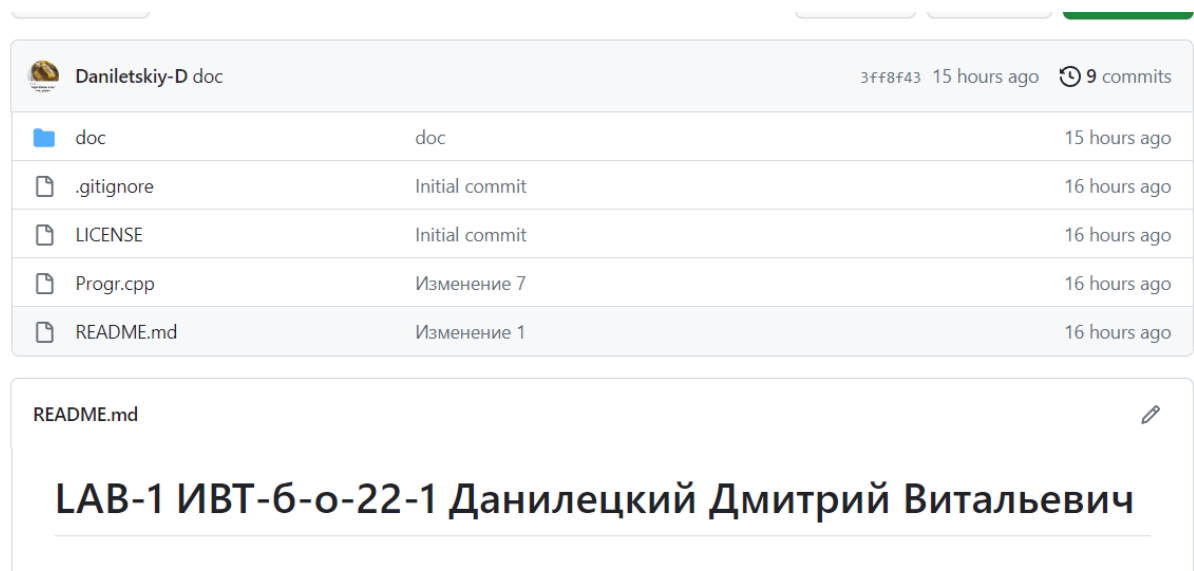


Рисунок 7. Подгруженные файлы на Github

### 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?

Недостатки локальных систем контроля версий (СКВ):

1) Ограниченный доступ: локальные СКВ ограничены работой на одной машине, что делает совместную работу и резервное копирование сложным;

2) отсутствие централизации: нет центрального сервера для хранения и управления версиями, что затрудняет координацию и отслеживание изменений;

3) ограниченная защита данных: потеря или повреждение локального репозитория может привести к потере данных.

Недостатки централизованных систем контроля версий (СКВ):

1) Одна точка отказа: если центральный сервер выходит из строя, весь процесс контроля версий может быть парализован;

2) зависимость от сети: работа с централизованными СКВ требует постоянного доступа к сети;

3) ограниченная гибкость: некоторые централизованные СКВ могут ограничивать способы ветвления и слияния кода.

3. К какой СКВ относится Git?

Git относится к распределенным системам контроля версий.

4. В чем концептуальное отличие Git от других СКВ?

Git отличается тем, что это распределенная система контроля версий (РСКВ), где каждая копия репозитория содержит всю историю, но может работать независимо и совместно с другими копиями. Также Git представляет свои данные как, скажем, поток снимков. Это обеспечивает гибкость и надежность в управлении версиями кода.

5. Как обеспечивается целостность хранимых данных в Git?

Целостность хранимых данных в Git обеспечивается с использованием хэш-суммы SHA-1 для каждого объекта в репозитории. Каждый коммит, файл и структура данных имеют свою уникальную хэш-сумму, и даже небольшие изменения в данных приводят к изменению хэша. Это позволяет обнаруживать любые изменения или повреждения данных, сохраняя их целостность.

6. В каких состояниях могут находиться файлы в Git? Как связаны эти состояния?

В Git файлы могут находиться в трех состояниях:

- 1) Modified (Измененные): Файлы были изменены в вашем рабочем каталоге;
- 2) Staged (Индексированные): Изменения в файлах были добавлены в индекс и готовы к фиксации (commit);
- 3) Committed (Зафиксированные): Изменения были сохранены в репозитории после коммита.

Связь между этими состояниями заключается в том, что изменения сначала происходят в рабочем каталоге, затем добавляются в индекс, а затем фиксируются в репозитории коммитом. Этот процесс позволяет контролировать версии и управлять изменениями в Git.

7. Что такое профиль пользователя в GitHub?

Профиль пользователя в GitHub - это публичная страница, на которой отображается информация о пользователе, их активность, репозитории и вклад в проекты на GitHub.

8. Какие бывают репозитории в GitHub?

1. Публичные (Public)\*\*: Доступные для всех пользователей GitHub. Информация и код в них видны всем.
2. Приватные (Private)\*\*: Доступны только для конкретных пользователей или команды, которым предоставлен доступ. Информация и код в них остаются невидимыми для остальных пользователей GitHub.

9. Укажите основные этапы модели работы с GitHub.

- 1) Создание репозитория.
- 2) Добавление, фиксация и отправка изменений.
- 3) Получение и слияние изменений.
- 4) Ветвление и слияние кода.
- 5) Запросы на слияние.
- 6) Управление задачами и релизами.



7) Коллаборация и управление доступом.

10. Как осуществляется первоначальная настройка Git после установки?

Установка имени пользователя и адреса электронной почты: Эти данные будут ассоциированы с вашими коммитами.

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

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

11. Опишите этапы создания репозитория в GitHub.

- 1) Войдите в аккаунт на GitHub.
- 2) Нажмите "+" и выберите "New repository".
- 3) Заполните имя, описание и видимость.
- 4) Опционально, добавьте README и выберите лицензию.
- 5) Нажмите "Create repository".

12. Какие типы лицензий поддерживаются GitHub при создании репозитория?

MIT License: Простая пермиссивная лицензия, позволяющая свободное использование, модификацию и распространение вашего кода с минимальными ограничениями.

GNU General Public License v3.0: Свободная и открытая лицензия, обеспечивающая свободу использования, модификации и распространения кода, но с требованием, что производные работы также должны быть открытыми и свободными.

Apache License 2.0: Лицензия с открытым исходным кодом, позволяющая свободное использование, модификацию и распространение кода, но с определенными условиями и гарантиями.

GNU GPLv2: Лицензия с открытым исходным кодом, аналогичная GPLv3, но с некоторыми различиями в требованиях к распространению кода.

BSD 3-Clause License: Пермиссивная лицензия, позволяющая использовать, модифицировать и распространять код, с минимальными ограничениями.

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

Клонирование репозитория GitHub осуществляется с помощью команды `git clone` в терминале, указывая URL репозитория. Клонирование позволяет скопировать удаленный репозиторий на локальный компьютер для работы над проектом. Это полезно для совместной разработки, внесения изменений и отслеживания версий кода без необходимости постоянно подключаться к серверу GitHub.

14. Как проверить состояние локального репозитория Git?

Для проверки состояния локального репозитория Git используется команда: `git status`. Эта команда покажет список измененных файлов, незакоммиченные изменения и другую информацию о текущем состоянии вашего репозитория.

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` ?

Добавление/изменение файла: Файл переходит из состояния "Untracked" в состояние "Unmodified" (неслеживаемый -> неизменный).

Добавление файла под версионный контроль с помощью `git add`: Файл переходит из состояния "Unmodified" в состояние "Staged" (неизменный -> проиндексированный).

Фиксация (коммит) изменений с помощью `git commit`: Все проиндексированные файлы становятся состоянием "Committed" (зафиксированный), и новый коммит создается.

Отправка изменений на сервер с помощью `git push`: Локальные коммиты отправляются на сервер, и удаленный репозиторий на сервере обновляется в соответствии с локальными изменениями.

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`.

Компьютер 1 (Первый компьютер):

```
git clone <URL_репозитория>
```

```
git add .
```

```
git commit -m "..."
```

```
git push origin <... >
```

Компьютер 2 (Второй компьютер):

```
git checkout <... >
```

```
git pull origin <... >
```

Теперь оба локальных репозитория будут в синхронизированном состоянии с удаленным репозиторием на GitHub.

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

GitHub:

1) Видимость репозитория: GitHub предоставляет бесплатные публичные репозитории и платные приватные репозитории.

2) Интеграция с CI/CD: Предоставляет GitHub Actions для настройки непрерывной интеграции и доставки.

3) Расширения и приложения: Широкий выбор сторонних интеграций доступен через GitHub Marketplace.

4) Сообщество и социальные функции: GitHub имеет активное сообщество разработчиков и обширные возможности для обсуждения, оценки и управления задачами.

5) GitHub Copilot: Интеллектуальный инструмент для кода, разработанный совместно с OpenAI, предоставляющий подсказки и автозамену кода.

GitLab:

1) Видимость репозитория: GitLab предоставляет бесплатные приватные репозитории, что может быть привлекательным для команд, требующих больше конфиденциальности.

2) Интеграция с CI/CD: В GitLab встроены инструменты для настройки CI/CD пайплайнов.

3) Самоуправляемый вариант: Вы можете установить GitLab на своем собственном сервере, что дает больший контроль и управление над вашими данными и настройками.

4) Создание задач: GitLab предлагает обширные возможности для управления задачами, включая встроенные трекеры и доски.

5) Поддержка DevOps: GitLab интегрирует инструменты для полного цикла разработки (DevOps), включая управление контейнерами и автоматизацию развертывания.

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.

Программные средства с GUI для Git:

1) GitHub Desktop: Простой и интуитивный. Операции Git выполняются с помощью кликов мышью.

2) GitKraken: Многофункциональный с поддержкой множества платформ.

3) SourceTree: Бесплатный GUI-клиент с поддержкой Git и Mercurial.

Операции Git с GitHub Desktop: Клонирование, коммит, push, pull - всё выполняется через интуитивный интерфейс.

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