

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»**  
**Вариант**

Выполнил:  
Говоров Егор Юрьевич  
2 курс, группа ИВТ-б-о-22-1,  
09.03.02 «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 \_\_\_\_\_

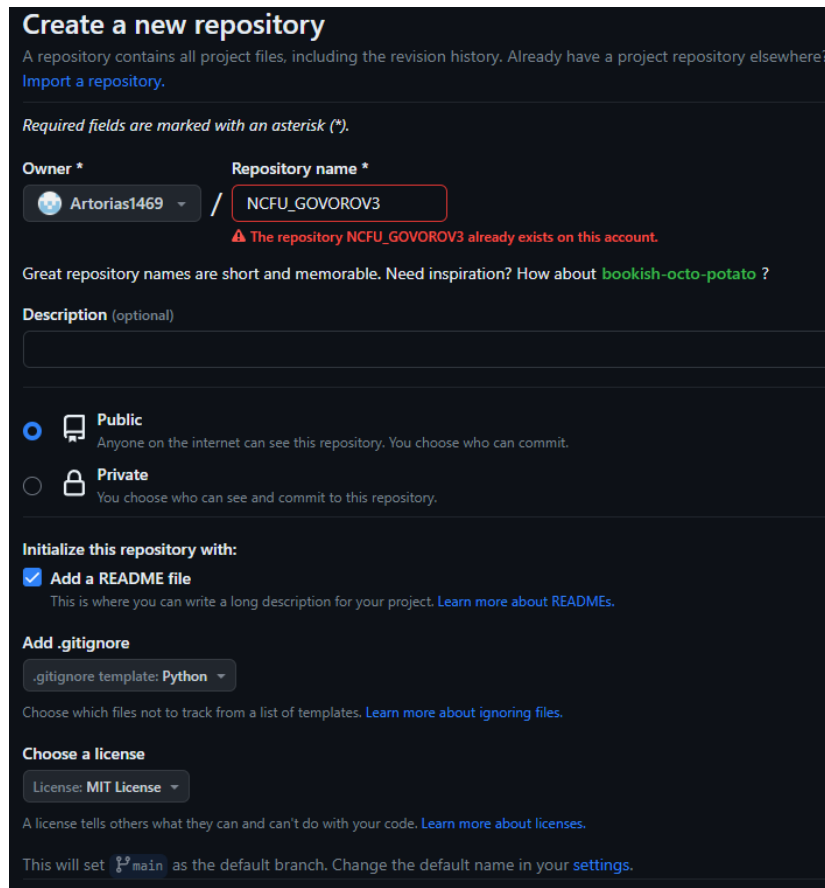
Ставрополь, 2023 г.

Тема: Исследование основных возможностей Git и GitHub

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

Ход работы:

### 1. 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 \* / Repository name \*

Artorias1469 / NCFU\_GOVOROV3

⚠ The repository NCFU\_GOVOROV3 already exists on this account.

Great repository names are short and memorable. Need inspiration? How about [bookish-octo-potato](#) ?

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: Python

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](#).

Рис 1. Создал репозиторий

### 2. Клонировал репозиторий в локальное хранилище

```
Admin@LAPTOP-ORH5MBF5 MINGW64 /d/Users/Admin/Рабочий стол/Git Repos_E
$ git clone https://github.com/Artorias1469/NCFU_GOVOROV3.git
Cloning into 'NCFU_GOVOROV3'...
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.
```

Рис 2. Клонирование в локальный репозиторий

### 3. В .gitignore добавил правилами для выбранного языка программирования и интегрированной среды разработки

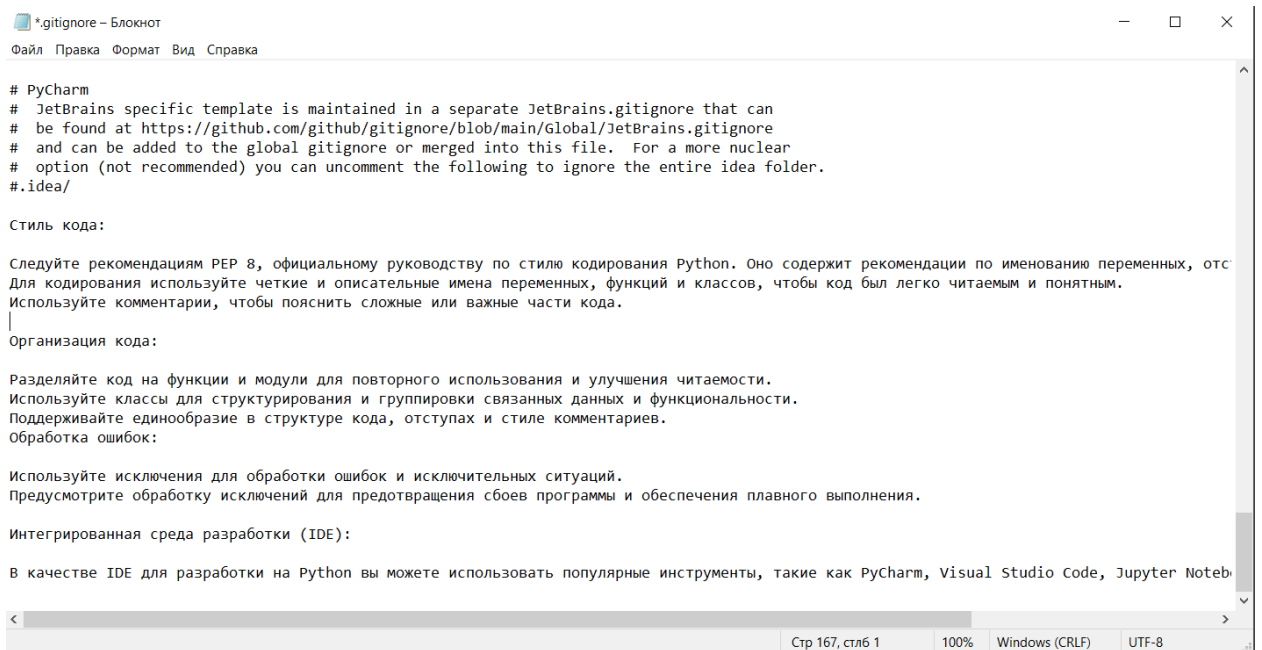


Рис 3. Правила пользования языком и средой

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

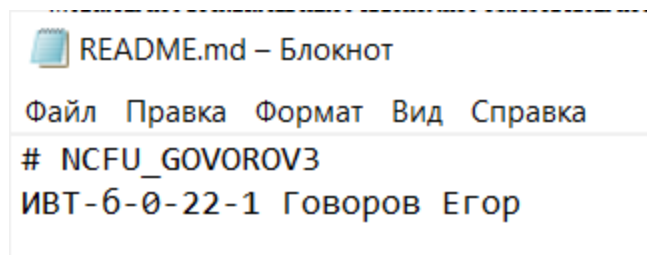


Рис 4. Добавил изменения в файл README

5. Написал небольшую программу

```
i = 1
while i > 2:
    i = i - 1
    if i < -3:
        i = i * -5
print("Егор")
print("hello world")
```

Рис 5. Программа на Python

6. Добавлял изменения в программа

```

OV3 (main)
$ git add .

Admin@LAPTOP-0RH5MBF5 MINGW64 /d/Users/Admin/Рабочий стол/Git Repos_E/NCFU_GOVOR
OV3 (main)
$ git push
Everything up-to-date

Admin@LAPTOP-0RH5MBF5 MINGW64 /d/Users/Admin/Рабочий стол/Git Repos_E/NCFU_GOVOR
OV3 (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   Pythone.py

Admin@LAPTOP-0RH5MBF5 MINGW64 /d/Users/Admin/Рабочий стол/Git Repos_E/NCFU_GOVOR
OV3 (main)
$ git commit -m "sdn"
[main 7efbba4] sdn
1 file changed, 2 insertions(+)

Admin@LAPTOP-0RH5MBF5 MINGW64 /d/Users/Admin/Рабочий стол/Git Repos_E/NCFU_GOVOR
OV3 (main)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 6 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 320 bytes | 320.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Artorias1469/NCFU_GOVOROV3.git
fb610c2..7efbba4 main -> main

Admin@LAPTOP-0RH5MBF5 MINGW64 /d/Users/Admin/Рабочий стол/Git Repos_E/NCFU_GOVOR
OV3 (main)
$ git add .

Admin@LAPTOP-0RH5MBF5 MINGW64 /d/Users/Admin/Рабочий стол/Git Repos_E/NCFU_GOVOR
OV3 (main)
$ git commit -m "sdn"
[main 5fcfcb6] sdn
1 file changed, 2 insertions(+), 1 deletion(-)

Admin@LAPTOP-0RH5MBF5 MINGW64 /d/Users/Admin/Рабочий стол/Git Repos_E/NCFU_GOVOR
OV3 (main)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 6 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 333 bytes | 333.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Artorias1469/NCFU_GOVOROV3.git
7efbba4..5fcfcb6 main -> main

```

Рис 6. Фиксирование изменений

```

Admin@LAPTOP-0RH5MBF5 MINGW64 /d/Users/Admin/Рабочий стол/Git Repos_E/NCFU_GOVOR
OV3 (main)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 6 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 1.13 KiB | 580.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Artorias1469/NCFU_GOVOROV3.git
5fcfcb6..ca12b79 main -> main

```

Рис 7. Отправка файлов на Github

## Ответы на вопросы:

1. СКВ - это сокращение от условно-контрольный вид. Она представляет собой систему классификации информации в рамках информационной безопасности. СКВ определяет, какие данные являются ограниченными и требуют особого внимания при обращении с ними.

Назначение СКВ заключается в обеспечении защиты конфиденциальной информации и контроля над доступом к ней. Она помогает понять, какие данные считаются секретными и требуют специальной обработки. СКВ также определяет различные уровни классификации информации, такие как "секретно", "очень секретно" и т.д.

Использование СКВ позволяет контролировать распространение и доступ к конфиденциальной информации, уменьшает риски утечки данных и улучшает общую информационную безопасность организации.

2. Ограниченная коллаборация: В локальных СКВ нет встроенных средств для совместной работы над проектами. Это затрудняет командную разработку, поскольку разработчики не могут легко обмениваться изменениями и комментариями к коду.

Сложное управление версиями: В локальных СКВ нет централизованной истории изменений, поэтому сложно отслеживать, кто и когда внес какие изменения. Это может привести к путанице и конфликтам при слиянии изменений.

Недостатки централизованных СКВ:

Одиночная точка отказа: В централизованных СКВ вся история проекта хранится на одном сервере. Если сервер выйдет из строя или будет недоступен, работа над проектом может быть затруднена или приостановлена.

Зависимость от доступности сети: Работа с централизованными СКВ требует постоянного доступа к серверу. Если разработчики не могут подключиться к серверу (например, из-за сетевых проблем), это может замедлить работу или привести к потере доступа к истории версий.

3. Git относится к категории распределенных систем контроля версий (Distributed Version Control System, DVCS). Это означает, что каждый участник проекта, работающий с Git, имеет полную копию репозитория со всей историей изменений локально на своем компьютере. Эта локальная копия репозитория позволяет разработчикам работать над проектом независимо, сохраняя возможность обмена изменениями между собой и с удаленным сервером. Git также поддерживает централизованные удаленные репозитории, но его основной принцип работы основан на распределенности.

4. Распределенная архитектура: Основное концептуальное отличие Git заключается в его распределенной архитектуре. Каждый участник проекта имеет полную копию репозитория, включая всю историю изменений, на своем локальном компьютере. Это позволяет разработчикам работать над проектом независимо и иметь возможность выполнения множества операций без необходимости постоянного подключения к центральному серверу. Такая архитектура делает Git более гибким и устойчивым к отказам.

Скорость и эффективность: Git разработан с учетом высокой производительности. Он обеспечивает быстрые операции коммитов, ветвления, слияния и переключения между ветками. Эффективные алгоритмы хранения данных и сжатия позволяют Git работать с большими репозиториями с высокой производительностью.

История изменений как набор снимков (snapshot): В Git каждый коммит представляет собой снимок всего состояния проекта на определенный момент времени, а не просто набор изменений. Это делает историю более понятной и позволяет легко перемещаться между версиями проекта.

Ветвление и слияние (branching and merging): Git делает ветвление и слияние очень простыми и эффективными операциями. Разработчики могут создавать новые ветки для разработки новых функций или исправления ошибок, а затем объединять их с основной веткой, минимизируя конфликты.

5. Хэш-функции: Git использует криптографические хэш-функции (как SHA-1) для вычисления хэш-сумм (хешей) для каждого объекта в репозитории. Эти хеши уникальны для содержимого объекта, и даже небольшое изменение в данных приведет к совершенно другому хешу. Это гарантирует, что данные не могут быть случайно или злонамеренно повреждены без обнаружения.

Объекты Git: Все файлы и история изменений в Git представлены в виде объектов. Существуют три основных типа объектов:

Blob (блоб): Объект, представляющий содержимое файла на определенный момент времени. Хэш-сумма Blob зависит от его содержимого.

Tree (дерево): Объект, представляющий каталог (папку) и его содержимое. Tree хранит хэши Blob и других Tree, что обеспечивает структуру файловой системы проекта.

Commit (коммит): Объект, который указывает на определенное состояние проекта в определенный момент времени. Коммит содержит хэши Tree и других коммитов, а также метаданные, такую как автора, дату и сообщение о коммите.

Цепочка коммитов: Все коммиты в Git образуют направленную ациклическую графовую структуру, где каждый коммит указывает на один или несколько родительских коммитов. Это позволяет отслеживать историю изменений.

6. Исходное состояние (Untracked): Файл находится в исходном состоянии, когда он не отслеживается Git. Это означает, что Git не знает о существовании файла и не участвует в его отслеживании или управлении.

Измененное состояние (Modified): Файл находится в измененном состоянии, когда его содержимое было изменено после последнего коммита. Git отслеживает эти изменения, и такие файлы могут быть включены в следующий коммит.

Индексированное состояние (Staged): Файл находится в индексированном состоянии, когда его изменения были добавлены в "индекс" (или "область подготовки", staging area). Индекс - это промежуточная зона, где вы выбираете, какие изменения пойдут в следующий коммит. Файлы в этом состоянии готовы к фиксации.

Зафиксированное состояние (Committed): Файл находится в зафиксированном состоянии, когда его изменения были сохранены в локальном репозитории в виде коммита. В этом состоянии файл считается без изменений относительно последнего коммита.

Связь между этими состояниями в Git следующая:

Когда вы создадите новый файл, он находится в исходном состоянии (Untracked).

После внесения изменений в файл, он переходит в измененное состояние (Modified).

Для включения изменений в следующий коммит, вы добавляете файл в индекс, переводя его в индексированное состояние (Staged).

Затем, при выполнении команды `git commit`, файл становится зафиксированным, и его состояние становится зафиксированным (Committed).

7. В GitHub профиль пользователя представляет собой публичную страницу, на которой отображается информация о пользователе и их активности на платформе. Профиль пользователя в GitHub предоставляет различную информацию, которая помогает другим пользователям понять их вклад в мир разработки программного обеспечения.

Имя и фотография: Пользователь может указать свое имя, псевдоним (username) и загрузить фотографию для отображения на странице профиля.



**Биография:** Здесь пользователь может кратко описать себя, свои интересы, профессиональный опыт и другую информацию, которая может быть интересна другим пользователям.

**Репозитории:** На странице профиля отображается список репозиторий, которые пользователь создал, в которых он участвует или которые он отметил звездой. Это позволяет другим пользователям узнать о проектах, над которыми работает данный пользователь.

**Активность:** Профиль пользователя также отображает их активность на GitHub. Это включает в себя информацию о коммитах, созданных запросах на слияние (Pull Requests), проблемах (Issues), обсуждениях и других действиях, совершенных на платформе.

**График активности:** На странице профиля есть график, который показывает активность пользователя в репозиториях по дням. Это может помочь определить, в какие дни пользователь активен и сколько времени уделяет работе над проектами.

**Подписчики и подписки:** Пользователь может видеть, кто подписан на его аккаунт (подписчики) и на кого он сам подписан (подписки).

**Организации:** Если пользователь является членом организаций на GitHub, то они также отображаются на его профиле.

8.

- Общедоступные репозитории (Public Repositories)
- Приватные репозитории (Private Repositories)
- Репозитории для организаций (Organization Repositories)
- Репозитории для пользователей (User Repositories)
- Форкнутые репозитории (Forked Repositories)
- Шаблоны репозиторий (Repository Templates)

9.

- Создание репозитория.
- Добавление, фиксация и отправка изменений.
- Получение и слияние изменений.

- Ветвление и слияние кода.
- Запросы на слияние.
- Управление задачами и релизами.
- Коллаборация и управление доступом.

10. 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.

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.

MIT License (Лицензия MIT)

GNU General Public License (GPL) (Общественная лицензия GNU общего пользования)

Apache License (Лицензия Apache)

Creative Commons Licenses (Лицензии Creative Commons)

Unlicense (Без лицензии)

Другие лицензии

12. 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.

13. с помощью `git status`

14. 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.

15. Клонирование репозитория на первом компьютере:

На первом компьютере выполните команду `git clone`, чтобы клонировать репозиторий с GitHub на ваш компьютер:

```
shell
```

```
git clone <URL репозитория>
```

Замените <URL репозитория> на URL вашего репозитория на GitHub.

Клонирование репозитория на втором компьютере:

На втором компьютере также выполните команду `git clone`, чтобы клонировать репозиторий с GitHub

```
shell
```

```
git clone <URL репозитория>
```

Это создаст копию репозитория на втором компьютере.

Работа с кодом:

На обоих компьютерах вы можете работать над кодом и вносить изменения в свои локальные репозитории.

Отправка изменений с первого компьютера на GitHub:

Если вы хотите отправить изменения с первого компьютера на GitHub, выполните следующие команды:

```
shell
```

`git add .` # Добавьте все измененные файлы в индекс

`git commit -m "Описание ваших изменений"` # Зафиксируйте изменения в коммите

`git push origin master` # Отправьте изменения на GitHub (здесь предполагается, что используется ветка master)

Замените "Описание ваших изменений" на описание ваших изменений.

Получение изменений с GitHub на втором компьютере:

Если вы хотите получить изменения с GitHub на втором компьютере, выполните следующую команду:

`shell`

`git pull origin master` # Получите изменения с GitHub (здесь предполагается, что используется ветка master)

Это обновит ваш локальный репозиторий на втором компьютере, включая все изменения, сделанные с первого компьютера и отправленные на GitHub.

## 16. GitLab и BitBucket

17. 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