

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ

Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций
Информационные технологии в телекоммуникациях

Отчет по лабораторной работе №1
«Исследование средств управления хостингом»

Выполнил студент группы ИТС-б-о-21-1

Крамаренко Илья Витальевич

« » _____ 20__ г.

Подпись студента _____

Проверил: Доцент, к.т.н, доцент кафедры
инфокоммуникаций

Воронкин А. В.

Работа защищена с оценкой: _____

(подпись)

Ставрополь, 2022

Лабораторная работа 1.1

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. Изучил теоретический материал работы.
2. Создал общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и выбранный мною язык программирования

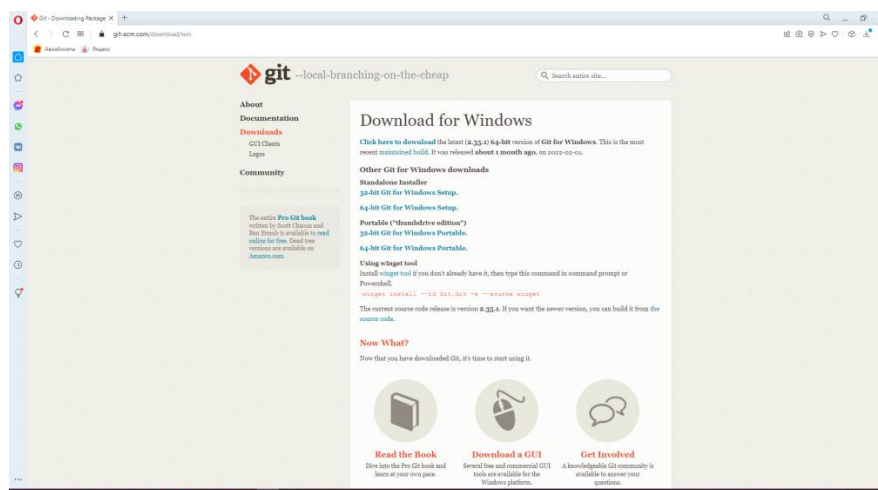


Рис 1. Выбираем Git для установки.

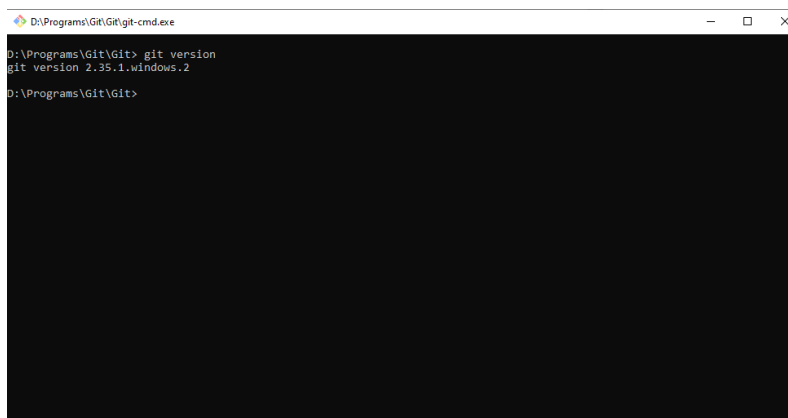


Рис 2. Проверка версии.

```
D:\Programs\Git\Git>git config --global user.name IliyaKr
D:\Programs\Git\Git>git config --global user.email iliyakr15@gmail.com
D:\Programs\Git\Git>
```

Рис 3. Настройка данных.

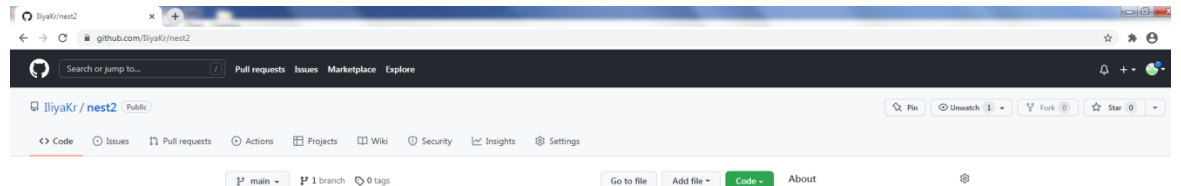


Рис 4. Создал новый репозиторий.

3. Выполнил клонирование созданного репозитория на рабочий компьютер.

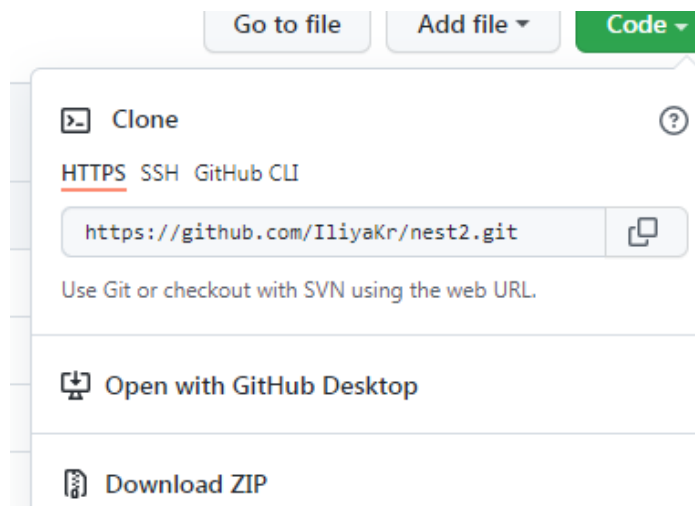


Рис 5. Получение адреса репозитория.

```
D:\Programs\Git\Git>git clone https://github.com/IliyaKr/nest2.git
Cloning into 'nest2'...
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.
D:\Programs\Git\Git>
```

Рис 6. Создал локальное хранилище проекта.

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

```
# 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
```

Рис 7. Описание игнорируемых файлов и папок.

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

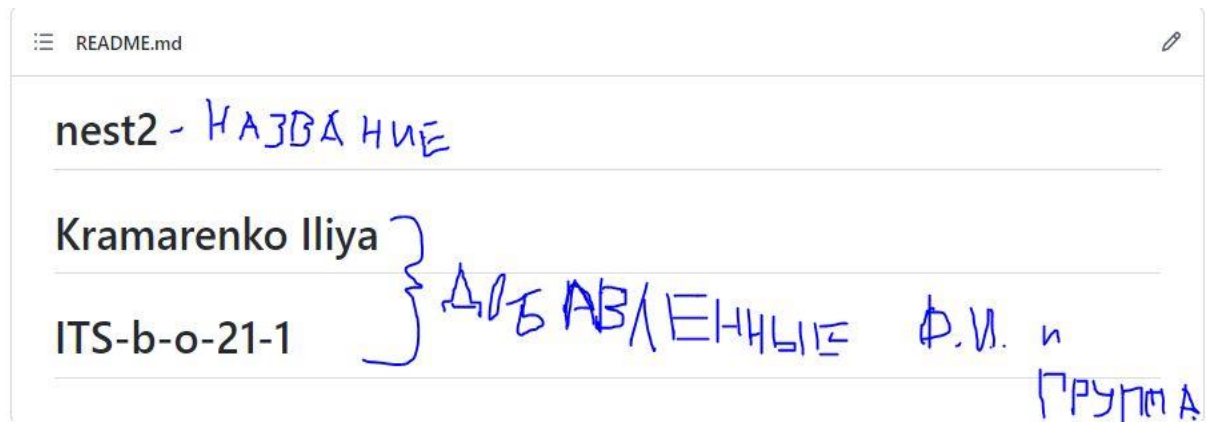


Рис 8. ФИО, группа.

6. Написал небольшую программу на выбранном мною языке программирования.

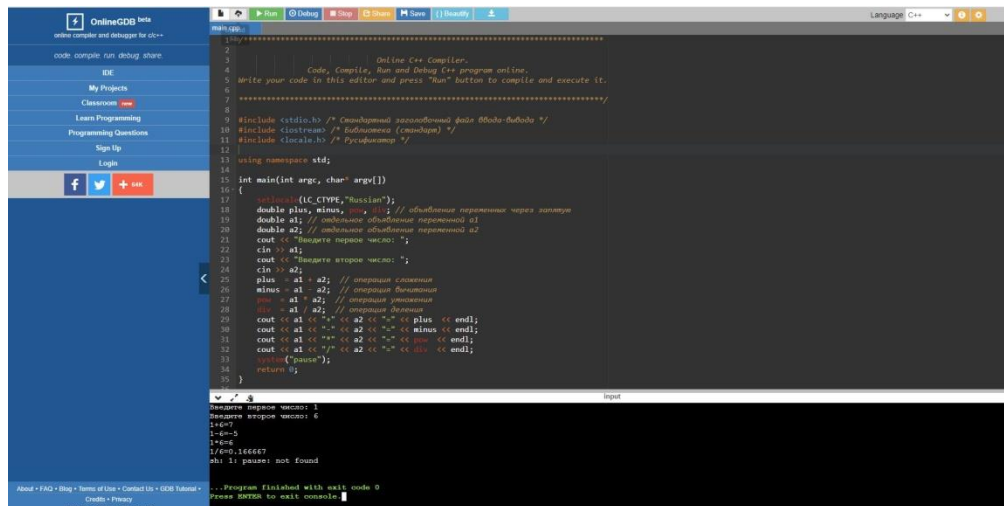


Рис 9. Написал программу

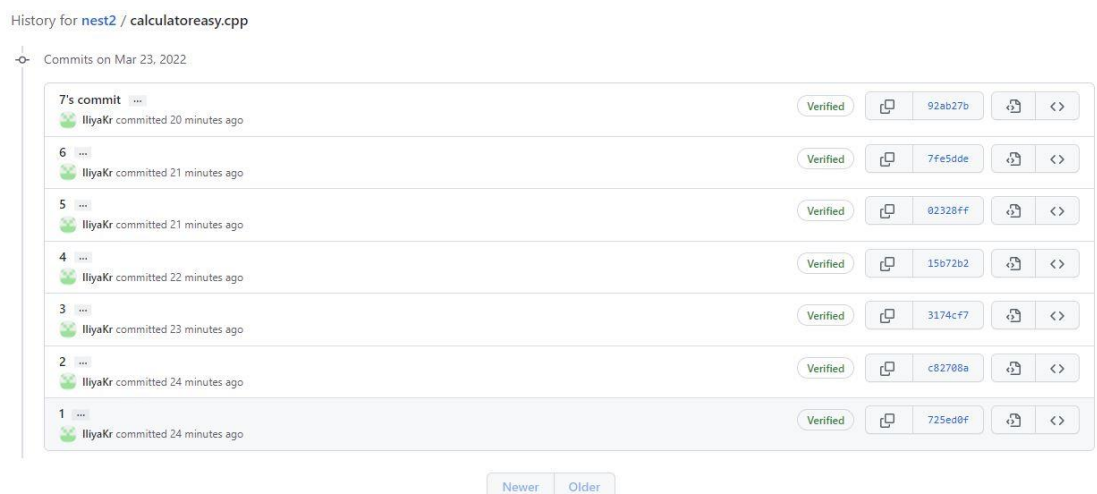


Рис 10. 7коммитов.

7. Добавил файл README и зафиксировал сделанные изменения.

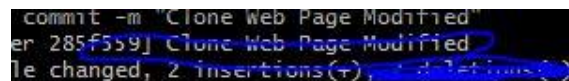


modified: README.md

Рис 11.

8. Добавил отчет по лабораторной работе в формате PDF в папку doc репозитория. Зафиксировал изменения.

9. Отправил изменения в локальном репозитории в удаленный репозиторий GitHub.



```
commit -m "Clone Web Page Modified"
[er 285f559] Clone Web Page Modified
1 file changed, 2 insertions(+), 2 deletions(-)
```

Рис 12.

10. Проконтролировал изменения, произошедшие в репозитории GitHub.



<https://github.com/hanishhazora805/ToolsQA.git>

Рис 13.

11. Отправил адрес репозитория GitHub на электронный адрес преподавателя.

Ответы на контрольные вопросы:

1) 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. Программисты обычно помещают в систему контроля версий исходные коды программ, но на самом деле под версионный контроль можно поместить файлы практически любого типа.

2) Недостатки

1. Отслеживает изменения только отдельных файлов, что не позволяет использовать ее для управления версиями больших проектов.

2. Не позволяет одновременно вносить изменения в один и тот же файл несколькими пользователями.

3. Низкая функциональность, по сравнению с современными системами контроля версий.

3) Git – это разновидность VCS (Version Control System). А VCS – это программа для работы с постоянно изменяющейся информацией. Такое ПО может хранить множество версий одного и того же файла (документа) и возвращаться к более раннему состоянию (версии).

4) 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. Эти системы (CVS, Subversion, Perforce, Bazaar и т. д.) представляют хранимую информацию в виде набора файлов и изменений, сделанных в каждом файле, по времени (обычно это называют контролем версий, основанным на различиях

5) 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

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

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

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

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

7) В профиле пользователя отображаем информацию о нем. Вы можете открыть свой профиль и внести изменения (например, вы можете добавить биографию или установить изображение). Создание аккаунта на GitHub ничем не отличается от создания аккаунтов на других сервисах подобного рода.

8) GitHub содержит миллионы проектов, написанных на разных языках программирования. Каждый проект размещается в своем собственном контейнере, который называется репозиторием. В нем можно хранить код, конфигурации, наборы данных, изображения и другие файлы, включенные в ваш проект. Любые изменения файлов в репозитории будут отслеживаться с помощью контроля версий. Если вы хотите найти какой-то конкретный репозиторий проекта, введите его имя или часть имени в поле поиска. Вы увидите список подходящих репозиториях

9) 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.

10) После установки GitHub нужно убедиться, что Git был успешно установлен, введите команду ниже в терминале, чтобы отобразить текущую версию вашего Git: (git version)

Если она сработала, то нужно добавить 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) 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 репозиториях, 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. Отлично, ваш репозиторий готов!

12) Сейчас, **при создании** нового **репозитория** вы можете указать параметры файла .gitignore и инициировать **создание** файла README. Кроме этого, **GitHub** предлагает вам выбрать **тип лицензии** вашего кода, такие как Apache, GPL, MIT и другие, не применяя **лицензию** LGPL автоматически, как это происходило раньше.

13) 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 <https://github.com/kushedow/flask-html>

14) 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) Следующие четыре команды предназначены для копирования файлов между рабочей директорией, сценой, также известной как «индекс», и историей, представленной в форме коммитов.

1. `git add` файлы копирует файлы в их текущем состоянии на сцену;
2. `git commit` сохраняет снимок сцены в виде коммита;
3. `git reset` — файлы восстанавливает файлы на сцене, а именно копирует файлы из последнего коммита на сцену. Используйте эту команду для отмены изменений, внесённых командой `git add` файлы. Вы также можете выполнить `git reset`, чтобы восстановить все файлы на сцене;
4. `git checkout` — файлы копирует файлы со сцены в рабочую директорию. Эту команду удобно использовать, чтобы сбросить нежелательные изменения в рабочей директории.

16) С одним репозиторием с разных компьютеров может работать несколько разработчиков или вы сами, если например работаете над одним и тем же проектом дома и на работе.

Для получения обновлений с удаленного репозитория воспользуйтесь командой: `git pull`

Если вы изменили ваши локальные файлы, то команда `git pull` выдаст ошибку. Если вы уверены, что хотите перезаписать локальные файлы, файлами из удаленного репозитория то выполните команды:

```
git fetch --all
```

```
git reset --hard github/master
```

Вместо `github` подставьте название вашего удаленного репозитория, которое вы зарегистрировали командой `git push -u`.

Как мы уже знаем, для того чтобы изменения выложить на удаленный репозиторий используется команда: `git push`

В случае, если в удаленном репозитории лежат файлы с версией более новой, чем у вас в локальном, то команда `git push` выдаст ошибку. Если вы уверены, что хотите перезаписать файлы в удаленном репозитории несмотря на конфликт версий, то воспользуйтесь командой: `git push -f`

Иногда возникает необходимость отложить ваши текущие изменения и поработать над файлами, которые находятся в удаленном репозитории. Для этого отложите текущие изменения командой: `git stash`

После выполнения этой команды ваша локальная директория будет содержать файлы такие же, как и при последнем коммите. Вы можете загрузить новые файлы из удаленного репозитория командой `git pull` и после этого вернуть ваши изменения которые вы отложили командой: `git stash pop`

17) Bitbucket — платформа, разработанная компанией Atlassian, которой также принадлежат инструменты для командной работы Jira, Trello и Confluence. Сервис интегрирован с этими и другими проектами — например, инструментами развёртывания приложений AWS CodeDeploy и Deploy to Azure, сервисом для поиска ошибок Instabug, средами разработки Visual Studio и Unity и прочими.

18) Tower это платный графический интерфейс Git для macOS и Windows. В настоящее время это один из ведущих профессиональных инструментов подобного типа. С его помощью вы сможете лучше познакомиться с системой контроля версий. Вам будут доступны в визуальном представлении все действия, которые можно совершать в Git. Сюда входит и разрешение конфликтов слияния, и совместная работа над проектами. Есть бесплатный пробный период.

Вывод: я исследовал основные базовые возможности системы контролей версий Git и веб сервис-хостинга для IT- проектов GitHub.