

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 ЛАБОРАТОРНОЙ РАБОТЕ №1
дисциплины «Программирование на Python»

Выполнил:
Кулешов Олег Иванович
2 курс, группа ИВТ-6-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.

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2023

Исследование основных возможностей Git и GitHub

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

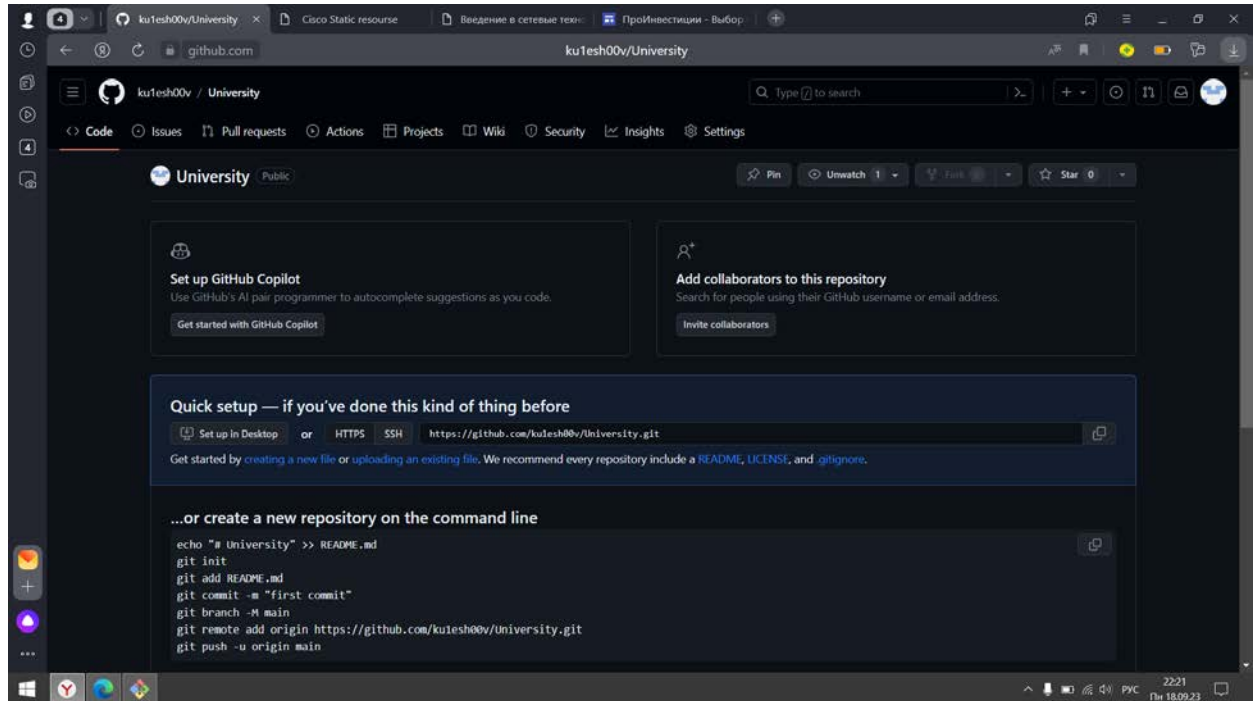


Рисунок 1. Создал репозиторий на GitHub

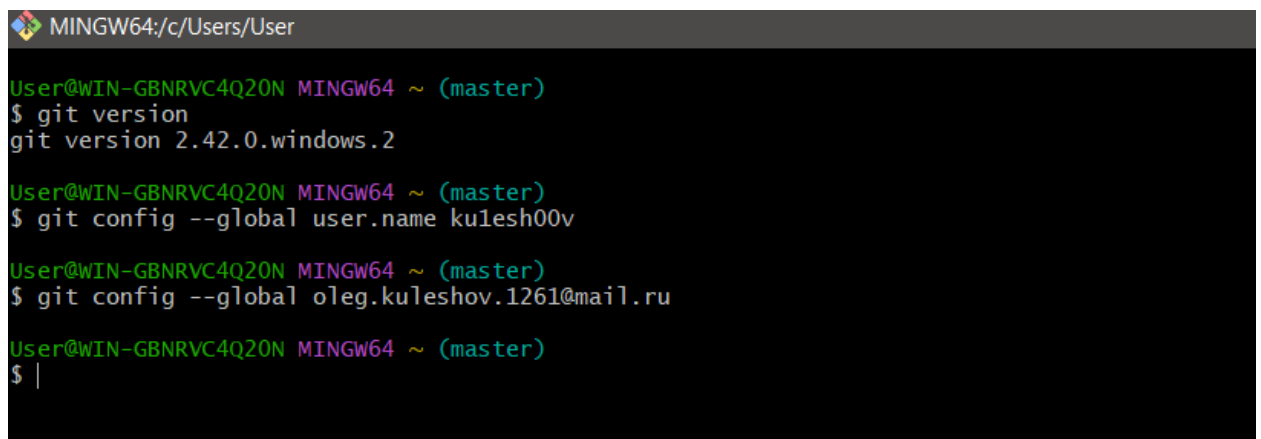


Рисунок 2. Проверил версию, ввёл имя и почту

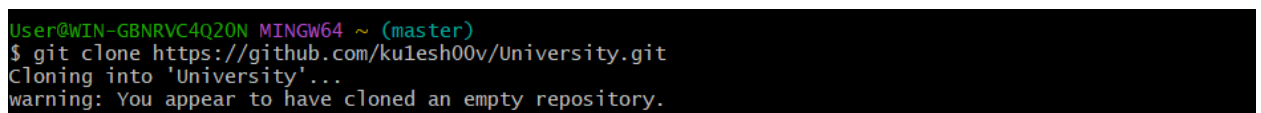


Рисунок 3. Клонировал репозиторий

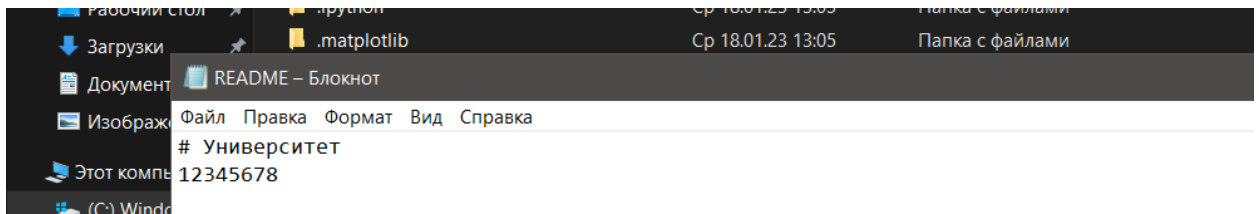


Рисунок 4. Изменил содержимое файла README.md

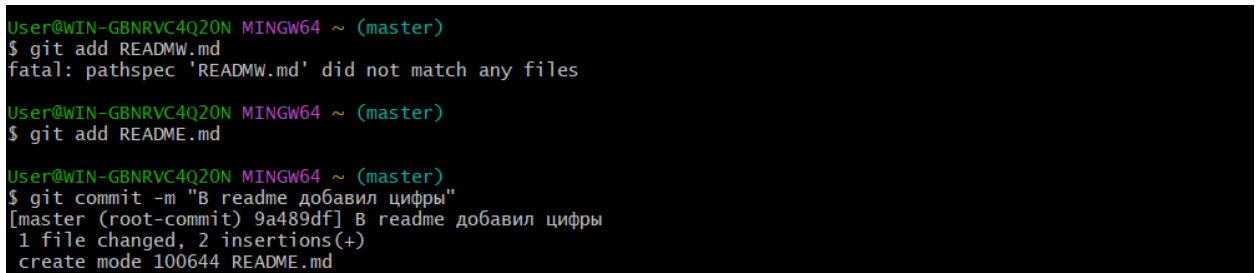


Рисунок 5. Добавил изменения в терминале и произвёл commit

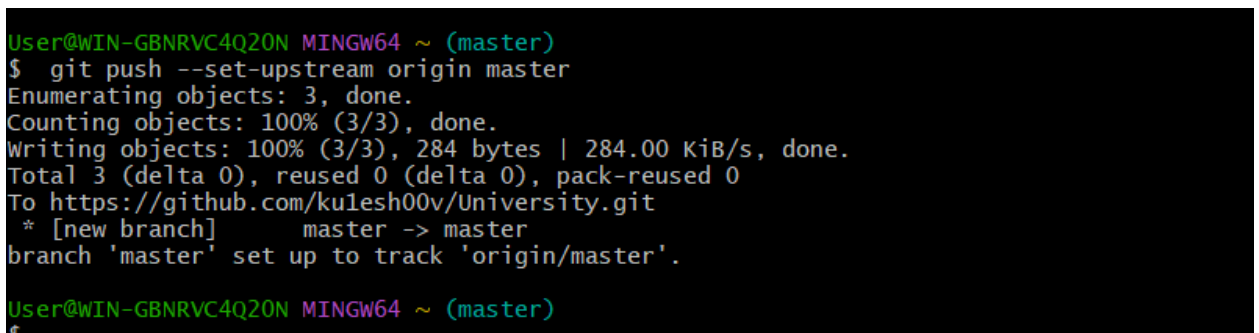


Рисунок 6. Произвёл push на GitHub

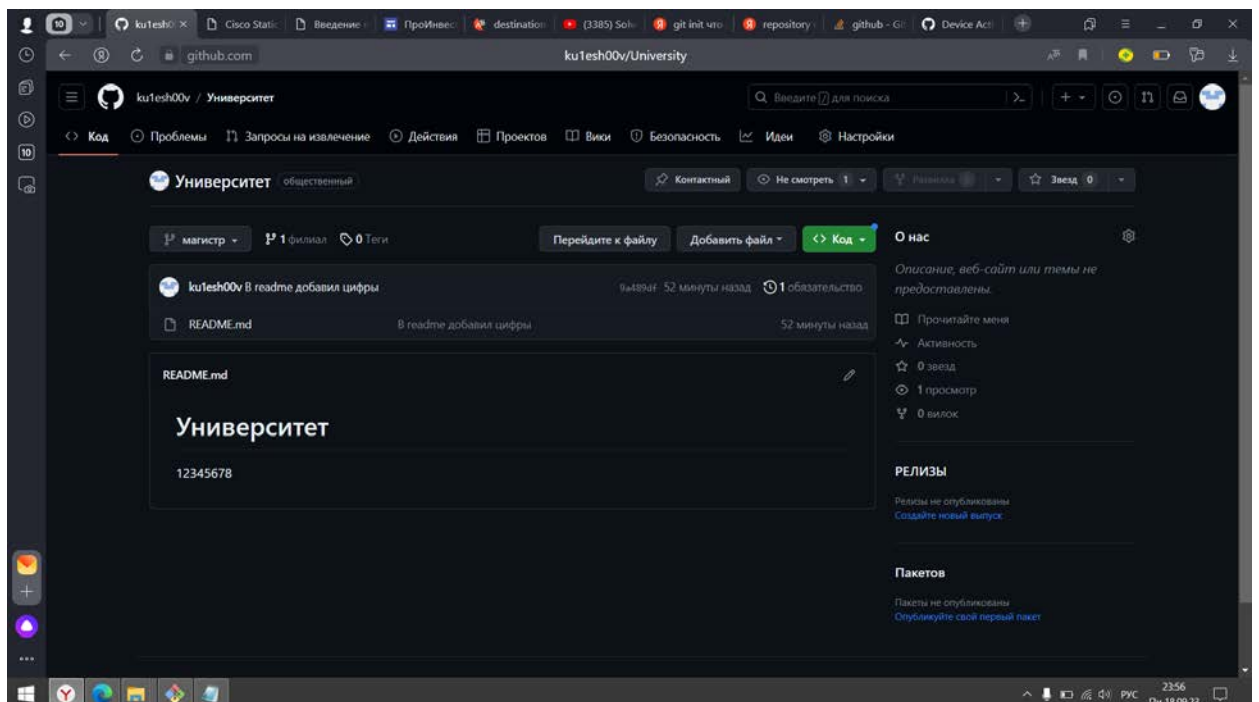


Рисунок 7. Проверил изменения в репозитории


```

User@WIN-GBNRVC4Q20N MINGW64 ~ (master)
$ git add README.md
warning: in the working copy of 'README.md', LF will be replaced by CRLF the next time Git touches it
User@WIN-GBNRVC4Q20N MINGW64 ~ (master)
$ git commit -m "Ввел данные о себе :)"
[master 51169d2] Ввел данные о себе :)
1 file changed, 1 insertion(+)
User@WIN-GBNRVC4Q20N MINGW64 ~ (master)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 342 bytes | 342.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/kulesh00v/university.git
0ea525d..51169d2 master -> master

```

Рисунок 11. Добавил изменения в терминал, произвёл commit и push

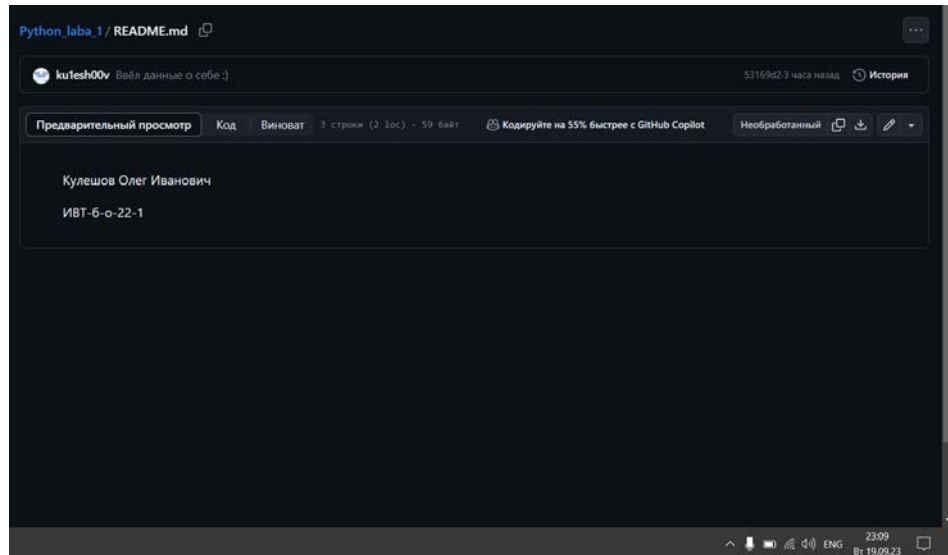


Рисунок 12. Результат махинаций с файлом README.md

```

kod.py
#Напишите программу, которая определяет, оканчивается ли год с данным номером на два нуля. Если год оканчивается, то выведите «YES», иначе выведите «NO».
#Формат входных данных: На вход программе подаётся натуральное число.
#Формат выходных данных: Программа должна вывести текст в соответствии с условием задачи.

x=int(input("Введите число:"))
if x%100==0:
    print("YES")
else:
    print("NO")

```

Рисунок 13. Написал небольшую программу

```

User@WIN-GBNRVC4Q20N MINGW64 ~ (master)
$ git add kod.py

User@WIN-GBNRVC4Q20N MINGW64 ~ (master)
$ git commit -m "добавил файл питон"
[master 693dce7] добавил файл питон
1 file changed, 10 insertions(+)
create mode 100644 kod.py

User@WIN-GBNRVC4Q20N MINGW64 ~ (master)
$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 631 bytes | 631.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/kulesh00v/University.git
ca3e483..693dce7 master -> master

```

Рисунок 14. Добавил файл с кодом программы, произвёл commit и push


<div>  kulesh00v Создать 123456 </div>		28ad95a 1 час назад	🕒 12 КОММИТОВ
📁 документ	Создать 123456		1 час назад
📄 .gitignore	внес готовый гитигнор		1 час назад
📄 README.md	Ввёл данные о себе :)		3 часа назад
📄 kod.py	добавил файл питон		2 часа назад

Рисунок 15. Результат в репозитории

Далее для удобства я переименовал свой репозиторий с University на Python_laba_1 для своего удобства. Затем ввёл в терминал следующую команду → `git remote set-url origin https://github.com/kulesh00v/Python_laba_1`

Ответы на 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.

2) Недостатки локальных СКВ:

1. Отсутствие централизации: В локальных СКВ каждый разработчик имеет свою копию репозитория на своем локальном компьютере. Это означает, что нет единого централизованного места для хранения и управления кодом. Это может привести к проблемам синхронизации и сложностям в обмене изменениями между разработчиками.

2. Отсутствие централизованной истории изменений: В локальных СКВ каждый разработчик имеет свою собственную историю изменений, которая не является доступной другим разработчикам. Это затрудняет отслеживание изменений, сравнение версий и управление конфликтами.

3. Ограниченные возможности совместной работы: Локальные СКВ не предоставляют механизмы для эффективной совместной работы над проектом. Разработчики могут столкнуться с проблемами синхронизации своих изменений и объединения их с изменениями других разработчиков.

Недостатки централизованных СКВ:

1. Одна точка отказа: Централизованные СКВ имеют единую точку отказа - сервер, на котором хранится репозиторий. Если сервер выходит из строя или возникают проблемы с доступом к нему, разработчики не могут получить доступ к коду или внести изменения.

2. Зависимость от сети: Централизованные СКВ требуют постоянного подключения к сети для доступа к репозиторию. Если разработчик находится в месте без доступа к сети, он не может получить доступ к коду или внести изменения.

3. Ограниченные возможности работы в оффлайн-режиме: Централизованные СКВ не предоставляют полноценной поддержки работы в оффлайн-режиме. Разработчики не могут вносить изменения или получать доступ к истории изменений без подключения к серверу.

4. Ограниченные возможности ветвления и слияния: Централизованные СКВ имеют ограниченные возможности ветвления и слияния кода. Это может привести к сложностям в управлении параллельными версиями проекта и объединении изменений разных разработчиков.

3) Git относится к распределенным системам контроля версий (РСКВ).

4) Концептуальное отличие Git от других систем контроля версий (СКВ) заключается в его распределенной архитектуре. В отличие от централизованных СКВ, таких как Subversion или CVS, где есть единственный центральный репозиторий, в Git каждый пользователь имеет полную копию всего репозитория на своем локальном компьютере.

5) Целостность хранимых данных в Git обеспечивается с помощью хэш-сумм, которые вычисляются для каждого объекта в репозитории. Каждый коммит, дерево файлов и содержимое файла имеют свой уникальный хэш-сумму, которая зависит от их содержимого.

6) Git файлы могут находиться в трех основных состояниях:

1. Измененное (Modified): Это состояние, когда файл был изменен, но еще не был проиндексирован для коммита. В этом состоянии Git отслеживает изменения в файле, но они не включены в следующий коммит.

2. Индексированное (Staged): После изменения файла, вы можете добавить его в индекс (stage) с помощью команды "git add". В этом состоянии Git записывает текущую версию файла в индекс, чтобы она была включена в следующий коммит.

3. Закоммиченное (Committed): Когда файл зафиксирован в репозитории с помощью команды "git commit", он считается закоммиченным. В этом состоянии Git сохраняет текущую версию файла в истории коммитов репозитория.

Связь между этими состояниями следующая:

- Изменения в файле происходят в состоянии "Измененное".
- После выполнения команды "git add" файл переходит в состояние "Индексированное".
- После выполнения команды "git commit" файл становится "Закоммиченным".

7) Профиль пользователя в GitHub является публичным и доступен для просмотра другим пользователям. Он представляет собой своеобразную визитную карточку, которая помогает другим пользователям понять, кто я такой и какую активность я проявляю на платформе.

8) В GitHub существуют различные типы репозиторий, включая:

1. Публичные репозитории: Это репозитории, которые видны всем пользователям GitHub. Любой пользователь может просматривать, клонировать и вносить изменения в публичные репозитории.
2. Приватные репозитории: Это репозитории, доступ к которым ограничен только для определенных пользователей или команд. Приватные репозитории не видны публично и обеспечивают более конфиденциальное хранение кода.
3. Форкнутые репозитории: Это копии публичных репозиторий, созданные другими пользователями. Форкнутые репозитории позволяют пользователям вносить изменения в код и отправлять запросы на слияние (pull requests) для внесения изменений в исходный репозиторий.
4. Шаблоны репозиторий: Это особый тип репозиторий, который предоставляет готовый набор файлов и структуру проекта для определенного типа приложений или библиотек. Шаблоны репозиторий могут быть использованы для быстрого создания новых проектов.
5. Организационные репозитории: Это репозитории, которые принадлежат организациям на GitHub. Организации могут создавать и управлять своими собственными репозиториями, приглашать пользователей и устанавливать различные уровни доступа.

Это только некоторые из типов репозиторий, которые могут быть найдены в GitHub. Каждый репозиторий имеет свои настройки доступа, функциональность и правила сотрудничества.

9) Основные этапы модели работы с GitHub включают:

1. Создание репозитория
2. Клонирование репозитория

3. Работа с кодом

4. Фиксация изменений: Пользователь фиксирует (commit) свои изменения в локальном репозитории с помощью команды `git commit`. В комментарии к фиксации указывается описание внесенных изменений.

5. Отправка изменений: Пользователь отправляет свои фиксации на удаленный репозиторий с помощью команды `git push`. Это обновляет удаленный репозиторий на GitHub.

6. Работа в команде: GitHub предоставляет возможность сотрудничества между разработчиками. Пользователи могут приглашать других пользователей к своим репозиториям, устанавливать права доступа и совместно работать над проектами.

7. Обновление и синхронизация: Пользователи могут получать обновления из удаленного репозитория с помощью команды `git pull`. Это позволяет поддерживать локальную версию проекта актуальной и синхронизированной с удаленным репозиторием.

10) Проверка версии: `git version`, указание имени пользователя: `git config –global user.name <<your name>>`, указание электронной почты пользователя: `git config –global user.email <<your email>>`, далее клонирование репозитория.

11) 1. Откройте веб-браузер и перейдите на сайт GitHub (<https://github.com>).

2. В правом верхнем углу страницы нажмите на кнопку "Sign in" и введите свои учетные данные для входа в свою учетную запись GitHub. Если у вас еще нет учетной записи GitHub, создайте ее, следуя инструкциям на сайте.

3. После успешного входа в свою учетную запись GitHub, нажмите на значок "+" в правом верхнем углу страницы и выберите "New repository" из выпадающего меню.

4. В открывшейся форме создания нового репозитория, заполните следующие поля:

- Repository name: Укажите название вашего репозитория.
- Description (optional): Добавьте описание к вашему репозиторию (необязательно).
- Public/Private: Выберите, будет ли ваш репозиторий публичным или приватным.
- Initialize this repository with a README: Создавать этот файл или нет, решать вам.

5. Нажмите на кнопку "Create repository", чтобы завершить создание репозитория.

12) 1. MIT License: Это очень перmissive лицензия, которая позволяет другим использовать, изменять и распространять ваш код без ограничений.

2. GNU General Public License (GPL): Эта лицензия требует, чтобы любые изменения, сделанные в вашем коде, также были распространены под GPL. Она обеспечивает свободу исследования, использования и изменения программного обеспечения.

3. Apache License: Эта лицензия также очень перmissive и позволяет другим использовать, изменять и распространять ваш код, но требует сохранения авторских прав и отказа от гарантий.

4. Creative Commons Licenses: GitHub также поддерживает различные версии лицензий Creative Commons, которые позволяют вам указывать условия использования вашего кода или контента.

Это только некоторые из множества лицензий, поддерживаемых GitHub.

13) Клонирование репозитория GitHub осуществляется с помощью команды "git clone" в терминале.

Нужно оно для получения локальной копии удаленного репозитория на компьютере.

14) При помощи команды git status

15) После добавления/изменения файла в локальный репозиторий Git, файл будет отображаться в состоянии "изменено" или "новый" при выполнении команды "git status". Это означает, что файл еще не добавлен в индекс (stage) для коммита.

После добавления нового/измененного файла под версионный контроль с помощью команды "git add", файл будет добавлен в индекс (stage) для коммита. Файл будет отображаться в состоянии "изменено" или "новый" при выполнении команды "git status", но теперь он будет готов к коммиту.

После фиксации изменений с помощью команды "git commit", изменения будут сохранены в локальном репозитории Git. Файлы, которые были добавлены в индекс (stage) с помощью команды "git add", будут включены в коммит. Коммит будет иметь уникальный идентификатор и сообщение, описывающее изменения.

После отправки изменений на сервер с помощью команды "git push", изменения будут загружены на удаленный репозиторий Git. Локальная ветка будет связана с соответствующей удаленной веткой, и все коммиты будут доступны другим участникам проекта.

16) 1. На первом компьютере выполните команду `git clone <URL репозитория>` для клонирования удаленного репозитория на ваш компьютер.

2. На втором компьютере выполните ту же команду `git clone <URL репозитория>`, чтобы склонировать удаленный репозиторий на второй компьютер.

3. Теперь есть два локальных репозитория, связанных с удаленным репозиторием на GitHub. Любые изменения, сделанные в одном из локальных репозиториях, не будут автоматически отображаться в другом.

4. Если вы внесли изменения в первом локальном репозитории и хотите синхронизировать их с удаленным репозиторием и вторым локальным репозиторием, выполните следующие команды:

- `git add .` - добавляет все измененные файлы в индекс для коммита.
- `git commit -m "Описание ваших изменений"` - фиксирует изменения в локальном репозитории.
- `git push origin <название ветки>` - отправляет изменения на удаленный репозиторий. <название ветки> - это название ветки, в которой вы хотите сохранить изменения.

5. На втором компьютере выполните команду `git pull origin <название ветки>`, чтобы загрузить последние изменения с удаленного репозитория и объединить их с вашим вторым локальным репозиторием. <название ветки> - это название ветки, которую вы хотите обновить.

17) Некоторые другие популярные сервисы, работающие с Git, включают в себя:

1. GitLab: GitLab предоставляет функциональность аналогичную GitHub, включая возможность хранения и управления репозиториями, контроль версий, задачи и запросы на слияние.

2. Bitbucket: Bitbucket является еще одним популярным сервисом хостинга репозиториях Git. Он предлагает множество функций, подобных GitHub и GitLab, включая возможность хранения и управления репозиториями, контроль версий, задачи и запросы на слияние.

Сравнение GitHub и GitLab:

Оба GitHub и GitLab являются популярными сервисами хостинга репозиториях Git, и оба предлагают множество схожих функций. Однако есть несколько отличий:

- Модель развертывания: GitHub предоставляет хостинг на своих серверах и не позволяет развернуть его на собственных серверах, в то время как GitLab

является open-source и может быть развернут на собственных серверах. Это дает большую гибкость и контроль над данными для пользователей GitLab.

- Непрерывная интеграция и развертывание: GitLab предлагает встроенную непрерывную интеграцию и развертывание (CI/CD), что упрощает автоматическую сборку, тестирование и развертывание приложений. GitHub предлагает поддержку CI/CD через интеграцию с другими инструментами, такими как Travis CI или CircleCI.

- Управление жизненным циклом приложений: GitLab предоставляет дополнительные инструменты для управления жизненным циклом приложений, такие как отслеживание ошибок и планирование проектов. GitHub сосредоточен в основном на хостинге репозитория и контроле версий.

18) Sourcetree: Sourcetree является бесплатным инструментом для работы с Git и Mercurial, разработанным компанией Atlassian. Он предоставляет графический интерфейс пользователя для выполнения операций Git, таких как клонирование репозитория, создание веток, выполнение коммитов, управление запросами на слияние и просмотр истории изменений. Sourcetree также предлагает интеграцию с другими инструментами Atlassian, такими как Jira и Bitbucket.

Пример реализации операций Git с помощью программного средства Sourcetree:

1. Клонирование репозитория: В Sourcetree вы можете нажать на кнопку "Клонировать/Добавить" и указать URL удаленного репозитория. Sourcetree автоматически склонирует репозиторий на ваш компьютер.

2. Создание ветки: Вы можете перейти во вкладку "Ветки" и нажать на кнопку "Создать ветку". Затем вы можете указать имя новой ветки и выбрать базовую ветку.

3. Выполнение коммитов: В Sourcetree вы можете просмотреть список измененных файлов и выбрать файлы, которые вы хотите включить в коммит. Затем вы можете написать сообщение коммита и выполнить коммит.

4. Управление запросами на слияние: Sourcetree предоставляет удобный интерфейс для создания и управления запросами на слияние. Вы можете перейти во вкладку "Запросы на слияние", нажать на кнопку "Создать запрос на слияние" и выбрать ветки для слияния.

5. Просмотр истории изменений: В Sourcetree вы можете просмотреть историю изменений репозитория, включая коммиты, ветки и запросы на слияние. Вы можете использовать фильтры и поиск, чтобы найти конкретные изменения.

