

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
«Основы кроссплатформенного программирования»
Вариант

Выполнил:
Митряшкина Дарина
2 курс, группа ИТС-б-о-23-1,
11.03.02 «Инфокоммуникационные
технологии и системы связи»,
очная форма обучения

(подпись)

Проверил:
Воронкин Р.А.

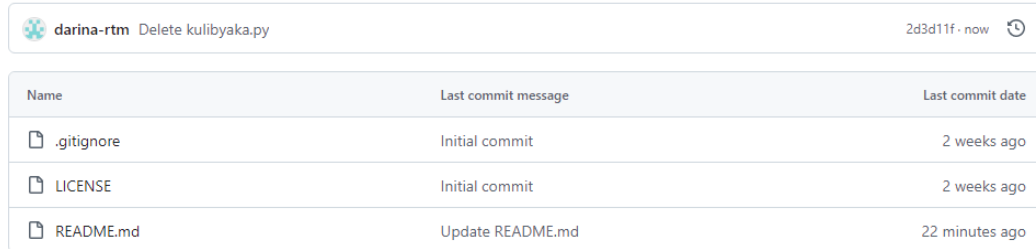
(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2024 г.

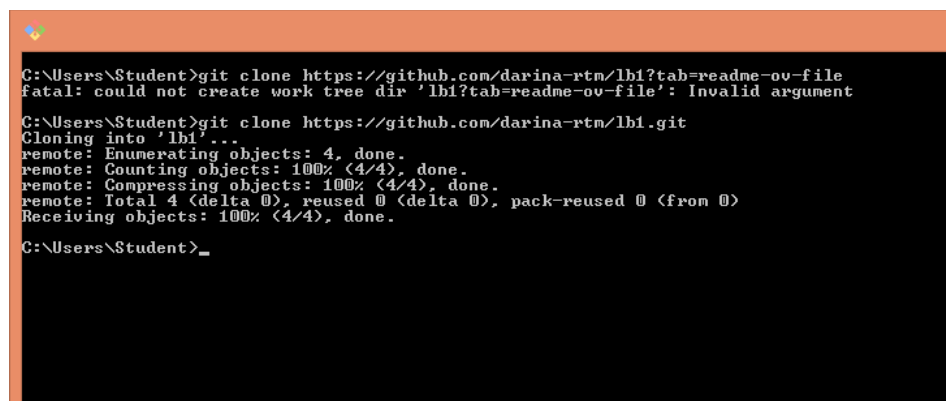
Лабораторная работа 1.1 Исследование основных возможностей Git и GitHub.

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



| darina-rtn Delete kulibyaka.py 2d3d11f · now | | |
|--|---------------------|------------------|
| Name | Last commit message | Last commit date |
| .gitignore | Initial commit | 2 weeks ago |
| LICENSE | Initial commit | 2 weeks ago |
| README.md | Update README.md | 22 minutes ago |

Рис.1. Создан общедоступный репозиторий на github



```
C:\Users\Student>git clone https://github.com/darina-rtn/lb1?tab=readme-ov-file
fatal: could not create work tree dir 'lb1?tab=readme-ov-file': Invalid argument
C:\Users\Student>git clone https://github.com/darina-rtn/lb1.git
Cloning into 'lb1'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (4/4), done.
C:\Users\Student>_
```

Рис.2. Выполнено клонирование созданного репозитория на рабочий компьютер

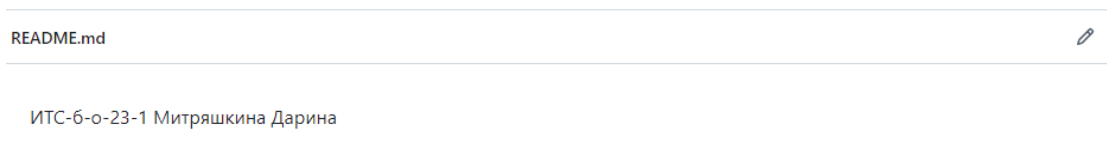


Рис.3. В файл README.md добавлена информацию о группе и ФИО студента, выполняющего лабораторную работу

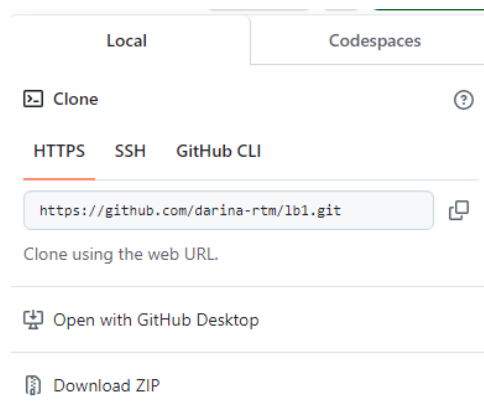


Рис.4. Выполнено клонирование репозитория

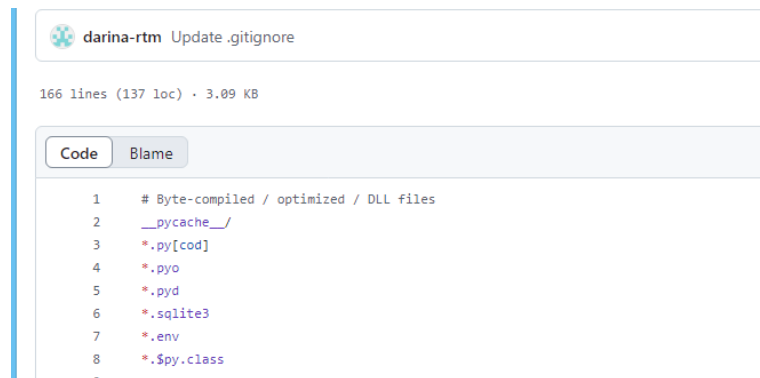


Рис.5. Файл “gitignore” дополнен необходимыми параметрами

```

1 def is_palindrome(text):
2     text = text.lower().replace(" ", "")
3     return text == text[::-1]
4
5 text = input("Введите текст: ")
6 if is_palindrome(text):
7     print("Это палиндром!")
8 else:
9     print("Это не палиндром.")

```

Рис.6. Написана небольшая программа

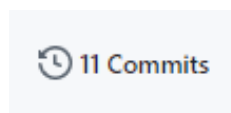


Рис.7. 11 коммитов

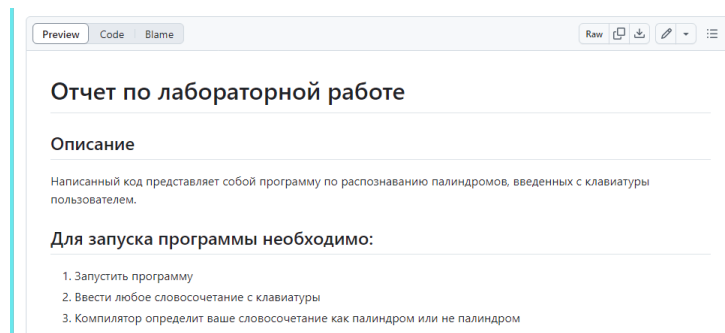


Рис.8. Добавлен файл README и внесены изменения

Ответы на контрольные вопросы:

1. Что такое СКВ и каково ее назначение?

Система управления версиями (также используется определение «система контроля версий», от англ. version control system, VCS или revision control system) — программное обеспечение для облегчения работы с изменяющейся информацией.

2. В чем недостатки локальных и централизованных СКВ?

Недостатки централизованных: отсутствие доступа к данным при сбое работы сервера; довольно низкая скорость работы (из-за возникновения сетевых задержек).

Недостатки локальных: возможность потери данных вследствие возникновения физических поломок оборудования; отсутствие возможности совместной разработки.

3. К какой СКВ относится Git?

Git относится к следующей системе контроля версий - распределенная.

4. В чем концептуальное отличие Git от других СКВ?

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

5. Как обеспечивается целостность хранимых данных в Git?

целостность может быть обеспечена за счет использования триггеров.

Триггер - это хранимая в БД процедура, исполняемая СУБД автоматически при удалении (DELETE), вставке (INSERT) или обновлении (UPDATE) записи.

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

Каждый файл в рабочем каталоге может находиться в одном из двух состояний: под версионным контролем (отслеживаемые) и нет (неотслеживаемые).

Отслеживаемые файлы — это те файлы, которые были в последнем снимке состояния проекта; они могут быть неизменёнными, изменёнными или подготовленными к коммиту.

Неотслеживаемые файлы — это всё остальное, любые файлы в вашем рабочем каталоге, которые не входили в ваш последний снимок состояния и не подготовлены к коммиту.

7. Что такое профиль пользователя в GitHub?

личная учетная запись — это наше удостоверение на GitHub.com и имеет имя пользователя и профиль.

8. Какие бывают репозитории в GitHub?

Локальный — расположен на одном компьютере, и работать с ним может только один человек.

Централизованный — расположен на сервере, куда имеют доступ сразу несколько программистов.

Распределенный — самый удобный вариант с облачным хранилищем.

9. Укажите основные этапы модели работы с GitHub.

- Создание репозитория
- Создание нового репозитория или получение его по существующему URL-адресу
- Операции с файлами
- Перемещение и удаление версий файлов репозитория
- Игнорирование некоторых файлов
- Исключение временных и вторичных файлов и директорий
- Сохранение фрагментов
- Сохранение и восстановление незавершённых изменений
- Внесение изменений
- Просмотр изменений и создание коммитов (фиксация изменений)
- Коллективная работа
- Именованные серии коммитов и соединение результатов работы

- Просмотр истории
- Просмотр и изучение истории изменений файлов проекта
- Откат коммитов
- Удаление ошибок и корректировка созданной истории
- Синхронизация с удалённым репозиторием
- Регистрация удалённого репозитория и обмен изменениями

10. Как осуществляется первоначальная настройка Git после установки?

Первоначальная настройка:

Настройка информации о пользователе для всех локальных репозиториев.

```
git config --global user.name "[имя]"
```

Устанавливает имя, которое будет отображаться в поле автора у выполняемых вами коммитов.

```
git config --global user.email "[адрес электронной почты]"
```

Устанавливает адрес электронной почты, который будет отображаться в информации о выполняемых вами коммитах.

11. Опишите этапы создания репозитория в GitHub.

В меню Git выберите "Создать репозиторий Git". В диалоговом окне "Создание репозитория Git" в разделе "Отправить" в новый удаленный раздел выберите GitHub. В разделе "Создание репозитория GitHub" диалогового окна "Создание репозитория Git" введите имя репозитория, который вы хотите создать

12. Какие типы лицензий поддерживаются GitHub при создании репозитория?

Поддерживаются следующие лицензии: MIT, Apache, GPL.

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

Откройте браузер и перейдите к учетной записи GitHub, перейдите на вкладку репозиториев и выберите репозиторий для клонирования. На странице репозитория GitHub выберите "Код", чтобы запустить всплывающее

окно клонирования. Скопируйте URL-адрес клона из всплывающего окна Clone.

Клонирование репозитория извлекает полную копию всех данных репозитория, которые есть у GitHub.com на тот момент времени, включая все версии каждого файла и папки для проекта.

14. Как проверить состояние локального репозитория Git?

Чтобы проверить текущее состояние репозитория, используем следующую команду `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`?

А. Добавление/изменение файла в локальный репозиторий Git:

Состояние: Файл становится "отслеживаемым" (tracked), но не находится под версионным контролем.

Как проверить: `git status` покажет файл в секции "Changes not staged for commit".

В. Добавление нового/измененного файла под версионный контроль с помощью команды `git add`:

Состояние: Файл переходит из "отслеживаемого" в "зафиксированное для коммита" (staged). Его изменения готовы к фиксации.

Как проверить: `git status` покажет файл в секции "Changes to be committed".

С. Фиксация (коммита) изменений с помощью команды `git commit`:

Состояние: Изменения, которые были зафиксированы (`git add`), добавляются в историю репозитория с комментарием, указанным в `git commit`. Файл становится "зафиксированным" (committed).

Как проверить: `git log` покажет историю коммитов, а `git status` будет показывать, что у вас нет изменений для коммита.

D. Отправка изменений на сервер с помощью команды `git push`:

Состояние: Изменения, зафиксированные в локальном репозитории, отправляются на удаленный сервер (например, GitHub).

Как проверить: `git status` покажет, что у вас нет изменений для коммита, а `git log` будет синхронизирован с историей коммитов на удаленном сервере.

Важно отметить:

Рабочая директория (`working directory`): Это текущее состояние ваших файлов на компьютере.

Индекс (`index`): Это временное хранилище, которое содержит информацию о файлах, которые будут зафиксированы в следующем коммите.

Локальный репозиторий (`local repository`): Это ваш локальный Git-репозиторий, который хранит историю изменений файлов.

Удаленный репозиторий (`remote repository`): Это копия вашего локального репозитория, хранящаяся на сервере.

Краткий итог:

...

1. Изменение файла: File > Untracked
2. `git add`: File > Staged
3. `git commit`: File > Committed
4. `git push`: Локальная история > Удаленная история

...

Дополнительные нюансы:

- `git add`-А добавляет все изменения в индекс.
- `git commit-am "Комментарий"` добавляет все изменения в индекс и коммитит их с комментарием.
- `git push origin master` отправляет изменения в ветку `master` на удаленный репозиторий `origin`.

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`.

Синхронизация двух локальных репозиториев с GitHub

А. Клонирование репозитория на первый компьютер:

```
git clone <адрес_репозитория_на_GitHub>
```

Результат: создается локальная копия репозитория на первом компьютере.

Б. Клонирование репозитория на второй компьютер:

```
git clone <адрес_репозитория_на_GitHub>
```

Результат: создается локальная копия репозитория на втором компьютере.

В. Настройка удаленного репозитория (origin) на обоих компьютерах:

На первом компьютере:

```
cd project-name
```

```
git remote add origin <адрес_репозитория_на_GitHub>
```

На втором компьютере:

```
cd project-name
```

```
git remote add origin <адрес_репозитория_на_GitHub>
```

Результат: оба локальных репозитория настроены на работу с удаленным репозиторием на GitHub.

Г. Синхронизация первого компьютера с удаленным репозиторием:

На первом компьютере:

```
git pull origin master
```

Результат: локальный репозиторий на первом компьютере получает последние изменения с удаленного репозитория.

Д. Синхронизация второго компьютера с удаленным репозиторием:

На втором компьютере:

```
git pull origin master
```

Результат: локальный репозиторий на втором компьютере получает последние изменения с удаленного репозитория.

Е. Работа с проектом на обоих компьютерах:

На первом компьютере: внесите изменения в файлы проекта и закоммитить их:

```
git add
```

```
git commit -m "Комментарий к изменениям"
```

```
git push origin master
```

На втором компьютере: внести изменения в файлы проекта и закоммитить их:

```
git add
```

```
git commit -m "Комментарий к изменениям"
```

```
git push origin master
```

Результат: изменения на обоих компьютерах будут отправлены на удаленный репозиторий.

Ж. Синхронизация после изменений:

На обоих компьютерах: перед тем, как начинать работу над новым набором изменений, обновить локальные репозитории:

```
git pull origin master
```

Результат: оба локальных репозитория будут синхронизированы с последними изменениями на удаленном репозитории.

В результате описанной последовательности действий, оба локальных репозитория будут синхронизированы с удаленным репозиторием на GitHub, что позволит эффективно работать над проектом с разных компьютеров.

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

Сервисы, работающие с Git:

GitLab:

- * Бесплатные публичные и частные репозитории: GitLab предлагает бесплатный план с неограниченным количеством частных репозиторий, что делает его привлекательным для команд, которые хотят сохранить свой код конфиденциальным.

- * Интеграция CI/CD: GitLab имеет встроенную систему непрерывной интеграции и доставки (CI/CD), что делает его более привлекательным для разработки программных продуктов.

- * Больше возможностей для управления проектами: GitLab предлагает более широкий набор инструментов для управления проектами (Kanban, Issue tracking), что делает его более функциональным для команд, работающих над большими проектами.

Bitbucket:

- * Специализация на работе с командами: Bitbucket ориентирован на работу с командами, в частности, с командами, использующими Atlassian Jira.

- * Интеграция с Jira: Bitbucket имеет глубокую интеграцию с Jira, что делает его более удобным для команд, использующих эту систему управления задачами.

Azure DevOps:

- * Интеграция с Azure: Azure DevOps тесно интегрирован с другими услугами Microsoft Azure, что делает его более привлекательным для разработчиков, работающих с этой платформой.

- * Широкие возможности CI/CD: Azure DevOps предлагает широкие возможности для CI/CD, в том числе поддержку различных языков программирования и платформ.

Launchpad:

* Open Source: Launchpad - это open-source платформа для разработки программного обеспечения, разработанная Canonical (компания, создавшая Ubuntu).

* Ориентация на open-source проекты: Launchpad часто используется для разработки open-source проектов, в том числе для Ubuntu и других дистрибутивов Linux.

Сравнение GitLab с GitHub:

GitHub:

Преимущества:

* Самый популярный сервис Git: GitHub является самым популярным сервисом Git, с большим количеством пользователей и проектов.

* Удобный интерфейс: GitHub имеет простой и интуитивно понятный интерфейс, который легко изучить.

* Широкая экосистема: GitHub имеет широкую экосистему инструментов и интеграций.

Недостатки:

* Ограничения бесплатного плана: Бесплатный план GitHub ограничен количеством частных репозиторий.

* CI/CD не является встроенной функцией: Для CI/CD нужно использовать дополнительные инструменты.

GitLab:

Преимущества:

* Бесплатные частные репозитории: GitLab предлагает бесплатный план с неограниченным количеством частных репозиторий.

* Встроенная CI/CD: GitLab имеет встроенную CI/CD систему.

* Дополнительные функции для управления проектами: GitLab предлагает более широкий набор инструментов для управления проектами.

Недостатки:

* Менее популярный сервис Git: GitLab менее популярен, чем GitHub, и имеет меньшее количество пользователей и проектов.

* Интерфейс может быть менее интуитивно понятным: интерфейс GitLab может быть менее интуитивно понятным для новичков.

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.

Графические интерфейсы для Git:

1. GitHub Desktop:

- Платформа: Windows, macOS
- Преимущества: Тесная интеграция с GitHub, простой и понятный интерфейс, удобен для начинающих.
- Недостатки: Ограниченная функциональность, не поддерживает GitLab и другие сервисы Git.

2. GitKraken:

- Платформа: Windows, macOS, Linux
- Преимущества: Мощный и функциональный интерфейс, поддерживает GitHub, GitLab и другие сервисы Git, широкий набор функций для визуализации и управления ветками.
- Недостатки: Платная программа, более сложный интерфейс для новичков.

3. Sourcetree:

- Платформа: Windows, macOS
- Преимущества: Бесплатная программа от Atlassian, поддерживает GitHub, Bitbucket и другие сервисы Git, интеграция с Jira и другими инструментами Atlassian.
- Недостатки: Может быть сложным для новичков, не поддерживает все функции Git из командной строки.

4. Git GUI:

- Платформа: Windows, macOS, Linux

- Преимущества: Бесплатная программа, open-source, доступна для многих операционных систем.

- Недостатки: Интерфейс более примитивный, чем у других программ, не поддерживает все функции Git из командной строки.

5. Tower:

- Платформа: macOS

- Преимущества: Мощный и функциональный интерфейс, поддерживает GitHub, GitLab и другие сервисы Git, удобен для профессиональных разработчиков.

- Недостатки: платная программа, дороговатая для новичков.

Реализация операций Git в GitKraken:

1. Клонирование репозитория:

- Нажмите кнопку "Clone" в левом верхнем углу.
- Введите URL репозитория и путь для клонирования.
- Нажмите "Clone".

2. Добавление файла в индекс:

- Выберите файл в дереве файлов.
- Нажмите кнопку "Stage Changes".

3. Создание коммита:

- Введите сообщение коммита.
- Нажмите кнопку "Commit".

4. Отправка изменений на удаленный репозиторий:

- Нажмите кнопку "Push".
- Выберите ветку и удаленный репозиторий.
- Нажмите "Push".

5. Получение изменений с удаленного репозитория:

- Нажмите кнопку "Pull".
- Выберите ветку и удаленный репозиторий.
- Нажмите "Pull".

6. Создание ветки:

- Нажмите кнопку "Branch" в левом верхнем углу.
- Введите название новой ветки.
- Нажмите "Create Branch".

7. Переключение на другую ветку:

- Выберите нужную ветку в списке веток.
- Нажмите кнопку "Checkout".

8. Слияние веток:

- Выберите ветку, которую нужно слить.
- Нажмите кнопку "Merge".
- Выберите ветку, в которую нужно слить.
- Нажмите "Merge".

9. Просмотр истории коммитов:

- Нажмите кнопку "History" в левом верхнем углу.
- Просмотрите историю коммитов в графическом виде.

GitKraken предоставляет более визуальный и интуитивный интерфейс для выполнения операций Git, что делает работу с системой версий более удобной и понятной. Он также предлагает множество дополнительных функций, например, визуализацию ветвей, инструменты для решения конфликтов и другие удобные возможности.

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