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 № 1.1**

**дисциплины**
**«Основы кроссплатформенного программирования»**

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

_____
(подпись)

Проверил:
Воронкин Р.А.
Доцент департамента цифровых, робототехнических систем и электроники

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2024 г.

# "Исследование основных возможностей Git и GitHub

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

## Порядок выполнения работы:

1. Изучил теоретический материал.
2. Создал общедоступный репозиторий на GitHub с MIT лицензией.
3. Выполнил клонирование репозитория на рабочий компьютер.

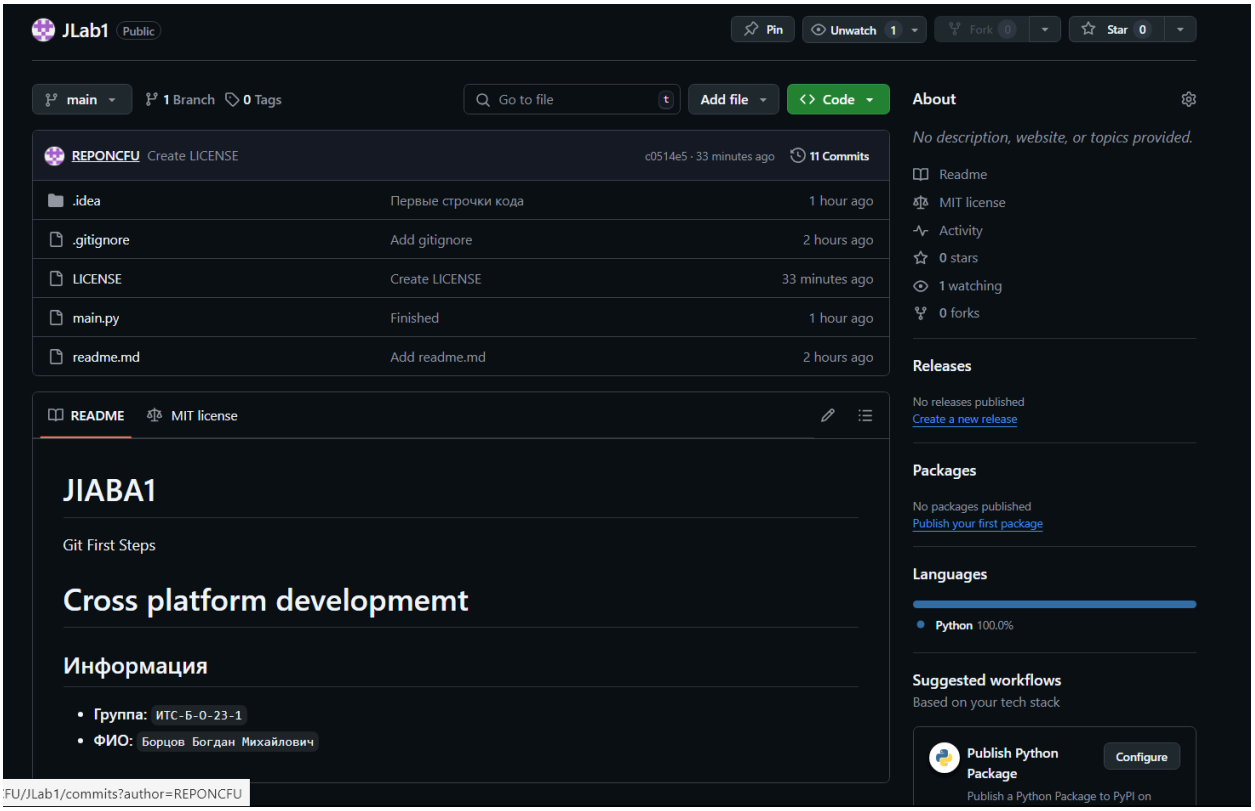


**Рис.1.** Созданный мной публичный репозиторий на GitHub

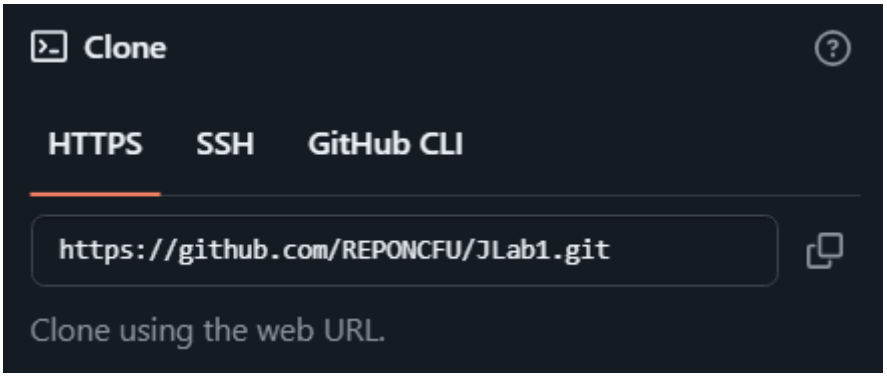


**Рис.2**. Ссылка на репозиторий GitHub

4. Создал и отредактировал файл .gitignore
5. Создал коммит и запушил
6. Создал и отредактировал файл readme.md
7. Создал коммит и запушил

```
C:\Users\Elony\JLab1>git add readme.md

C:\Users\Elony\JLab1>git add .

C:\Users\Elony\JLab1>git commit -m "Add readme.md"
[main (root-commit) b4a74c6] Add readme.md
 1 file changed, 9 insertions(+)
 create mode 100644 readme.md
```

**Рис.3**. Создал коммит

```
C:\Users\Elony\JLab1>git add .gitignore

C:\Users\Elony\JLab1>git add .

C:\Users\Elony\JLab1>git commit -m "Add gitignore"
[main fe7d596] Add gitignore
 1 file changed, 77 insertions(+)
 create mode 100644 .gitignore

C:\Users\Elony\JLab1>git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 1.02 KiB | 1.02 MiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/REPONCFU/JLab1.git
   b4a74c6..fe7d596  main -> main
```

**Рис.4**. Коммит и пуш

8. Создал файл main.py и в рамках нескольких коммитав доработал его до финального состояния

```
C:\Users\Elony\JLab1>git add main.py

C:\Users\Elony\JLab1>git add .

C:\Users\Elony\JLab1>git commit -m "Implement basic command responses"
[main 4757911] Implement basic command responses
 1 file changed, 7 insertions(+), 3 deletions(-)

C:\Users\Elony\JLab1>git add .

C:\Users\Elony\JLab1>git commit -m "Add random ML and AI jokes"
[main aacf6ea] Add random ML and AI jokes
 1 file changed, 13 insertions(+)

C:\Users\Elony\JLab1>git add .

C:\Users\Elony\JLab1>git commit -m "Implement funny action commands"
[main f068ea8] Implement funny action commands
 1 file changed, 5 insertions(+)

C:\Users\Elony\JLab1>git add .

C:\Users\Elony\JLab1>git commit -m "Handle invalid inputs and add exit functionality"
[main 214421d] Handle invalid inputs and add exit functionality
 1 file changed, 3 insertions(+)

C:\Users\Elony\JLab1>git add .
```

**Рис.5**. Добавление main.py и коммиты

9. Запушил все
10. Ответил на контрольные вопросы
11. Сформировал отчет

**Контрольные вопросы**

## 1. Что такое СКВ и каково ее назначение?

**СКВ (Система Контроля Версий)** — это программное обеспечение, предназначенное для управления изменениями в наборах файлов, таких как исходный код, документы и другие ресурсы. Основные назначения СКВ:

- **Отслеживание истории изменений**: Позволяет видеть, какие изменения были внесены, кем и когда.
- **Восстановление предыдущих версий**: Возможность отката к любой предыдущей версии файла или проекта.
- **Совместная работа**: Облегчает работу нескольких разработчиков над одним проектом, обеспечивая механизм слияния изменений.
- **Управление ветками**: Позволяет создавать параллельные линии разработки, такие как новые функции или исправления багов.
- **Безопасность и целостность данных**: Обеспечивает надежное хранение и защиту данных от потери или повреждения.

## 2. В чем недостатки локальных и централизованных СКВ?

**Недостатки локальных СКВ:**

1. **Отсутствие централизованного хранилища**: Изменения хранятся только на локальной машине, что усложняет совместную работу.
2. **Отсутствие истории изменений**: Могут быть ограничены или отсутствовать возможности отслеживания истории.
3. **Риск потери данных**: Отсутствие резервных копий увеличивает риск потери данных при сбое оборудования.

**Недостатки централизованных СКВ:**

1. **Единая точка отказа**: Если сервер СКВ становится недоступен, вся команда не может работать.
2. **Ограниченная производительность**: При большом количестве пользователей сервер может испытывать нагрузку.
3. **Меньшая гибкость**: Централизованные системы обычно менее гибкие в плане ветвления и слияния изменений.
4. **Зависимость от сети**: Требуется постоянное подключение к серверу для работы.

## 3. К какой СКВ относится Git?

**Git** относится к **распределенным системам контроля версий (Distributed Version Control Systems, DVCS)**. В Git каждый разработчик имеет полную копию репозитория, включая всю историю изменений, что позволяет работать автономно и обеспечивает высокую скорость операций.

## 4. В чем концептуальное отличие Git от других СКВ?

Концептуальное отличие Git от других СКВ заключается в том, что Git является **распределенной системой**. Это означает, что каждый разработчик имеет полную копию репозитория на своей локальной машине, включая всю историю изменений. В отличие от централизованных систем, где есть один центральный сервер, Git позволяет выполнять

большинство операций локально, обеспечивая большую гибкость, скорость и устойчивость к сбоям. Кроме того, Git использует эффективную модель ветвления и слияния, что упрощает работу с параллельными изменениями.

### 5. Как обеспечивается целостность хранимых данных в Git?

**Целостность данных в Git** обеспечивается с помощью **криптографических хешей SHA-1 (в новых версиях может использоваться SHA-256)**. Каждый объект в Git (коммит, дерево, блоб) идентифицируется уникальным хешем, который вычисляется на основе содержимого объекта. Это гарантирует, что любые изменения в содержимом объекта приведут к изменению его хеша, что позволяет обнаруживать любые попытки изменения данных. Кроме того, структура репозитория построена таким образом, что изменения в одном объекте влияют на связанные объекты, что дополнительно защищает целостность всей истории проекта.

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

В Git файлы могут находиться в следующих состояниях:

1. **Untracked (Неотслеживаемые)**: Файлы, которые находятся в рабочем каталоге, но еще не добавлены под контроль версий.
2. **Tracked (Отслеживаемые)**:
    - **Modified (Измененные)**: Файлы, которые были изменены после последнего коммита.
    - **Staged (Подготовленные)**: Измененные файлы, добавленные в индекс с помощью команды `git add` для включения в следующий коммит.
    - **Committed (Закоммиченные)**: Файлы, сохраненные в истории репозитория посредством команды `git commit`.

**Связь между состояниями:**

- **Untracked** → **Staged**: Добавление файла под контроль версий с помощью `git add`.
- **Modified** → **Staged**: Изменение отслеживаемого файла и его добавление в индекс.
- **Staged** → **Committed**: Создание коммита с подготовленными изменениями.
- После коммита файлы возвращаются в состояние **Tracked** и готовы к дальнейшим изменениям.

### 7. Что такое профиль пользователя в GitHub?

**Профиль пользователя в GitHub** — это личная страница, отображающая информацию о пользователе. В профиле содержатся:

- **Аватар и имя пользователя**.
- **Биография**: Краткое описание пользователя.
- **Контактная информация**: Ссылки на веб-сайт, социальные сети и т.д.
- **Список репозиториев**: Публичные и приватные репозитории, в которых пользователь участвует.
- **Активность**: Коммиты, пулл-запросы, Issues и другие действия.
- **Пиннерованные репозитории**: Выбранные пользователем репозитории, которые он хочет выделить на своем профиле.

Профиль позволяет другим пользователям узнать больше о владельце аккаунта, его проектах и вкладах в другие проекты.

### 8. Какие бывают репозитории в GitHub?

В GitHub существуют два основных типа репозиториев:

1. **Публичные репозитории (Public repositories)**:
    - Доступны для просмотра всеми пользователями GitHub.
    - Любой может клонировать, форкать и просматривать их содержимое.
    - Подходят для открытых проектов и сотрудничества с широким сообществом.
2. **Приватные репозитории (Private repositories)**:
    - Доступны только для выбранных пользователей или команд.
    - Их содержимое скрыто от остальных.
    - Подходят для разработки закрытых проектов, коммерческих приложений или проектов с ограниченным доступом.

Кроме того, репозитории могут быть:

- **Forked (форкнутые)**: Копии чужих репозиториев, позволяющие вносить изменения независимо от оригинала и предлагать свои изменения через пулл-запросы.
- **Archived (архивированные)**: Репозитории, которые больше не поддерживаются или используются, их состояние заморожено для предотвращения дальнейших изменений.

### 9. Укажите основные этапы модели работы с GitHub.

Основные этапы модели работы с GitHub:

1. **Создание аккаунта**: Регистрация на платформе GitHub.
2. **Создание репозитория**: Создание нового репозитория на GitHub, выбор между публичным и приватным.
3. **Клонирование репозитория**: Копирование репозитория на локальную машину с помощью команды `git clone`.
4. **Работа с файлами**: Внесение изменений, добавление новых файлов, редактирование существующих.
5. **Добавление изменений в индекс**: Использование команды `git add` для подготовки изменений к коммиту.
6. **Коммит изменений**: Сохранение изменений в локальный репозиторий с помощью команды `git commit`.
7. **Отправка изменений на GitHub**: Использование команды `git push` для отправки локальных коммитов в удаленный репозиторий на GitHub.
8. **Обновление локального репозитория**: Использование команды `git pull` для получения и интеграции изменений из удаленного репозитория.
9. **Работа с ветками**: Создание, переключение и слияние веток для организации разработки.
10. **Создание пулл-запросов (Pull Requests)**: Предложение изменений для объединения в основную ветку, обсуждение и ревью кода.
11. **Мердж изменений**: Объединение изменений из пулл-запросов после одобрения.
12. **Управление проблемами и задачами**: Использование системы Issues для отслеживания багов, задач и фич.

### 10. Как осуществляется первоначальная настройка Git после установки?

Первоначальная настройка Git после установки включает настройку основных параметров пользователя, которые будут использоваться при коммитах. Для этого выполняются следующие команды в терминале:

**Настройка имени пользователя**:
```
git config --global user.name "Ваше Имя"
```

**Настройка адреса электронной почты**:
```
git config --global user.email "ваш.email@example.com"
```

**Выбор редактора по умолчанию** (необязательно, по умолчанию используется Vim):
`git config --global core.editor "vim"`

**Настройка формата отображения диффов** (например, для цветного вывода):
`git config --global color.ui true`

**Проверка настроек**:
`git config --list`

Эти настройки сохраняются в глобальном конфигурационном файле Git и применяются ко всем репозиториям пользователя на данной машине.

### 11. Опишите этапы создания репозитория в GitHub.

Этапы создания репозитория в GitHub:

1. **Вход в аккаунт GitHub**:
   - Перейдите на GitHub и войдите в свой аккаунт.
2. **Создание нового репозитория**:
   - Нажмите на кнопку "+" в верхнем правом углу страницы и выберите "New repository" (Новый репозиторий).
3. **Заполнение информации о репозитории**:
   - **Имя репозитория**: Введите уникальное имя для вашего репозитория.
   - **Описание (необязательно)**: Кратко опишите назначение репозитория.
   - **Видимость**: Выберите между публичным (Public) и приватным (Private) репозиторием.
   - **Инициализация репозитория**: Можно добавить файл README, .gitignore, лицензию, выбрав соответствующие опции.
4. **Создание репозитория**:
   - Нажмите кнопку "Create repository" (Создать репозиторий).
5. **Настройка локального репозитория**:
   - Если вы инициализировали репозиторий с README, следуйте инструкциям на экране для клонирования репозитория на локальную машину или добавления удаленного репозитория к существующему локальному репозиторию.
6. **Добавление файлов и коммитов**:
   - Добавляйте файлы, вносите изменения, создавайте коммиты и отправляйте изменения на GitHub с помощью команд `git add`, `git commit` и `git push`.

### 12. Какие типы лицензий поддерживаются GitHub при создании репозитория?

GitHub поддерживает широкий выбор лицензий при создании репозитория. Некоторые из наиболее распространенных типов лицензий включают:

1. **MIT License**: Простая и разрешительная лицензия, позволяющая свободное использование, копирование, модификацию и распространение.
2. **GNU General Public License (GPL)**: Строгая лицензия с требованием распространять производные работы под той же лицензией.
3. **Apache License 2.0**: Разрешает использование, модификацию и распространение с дополнительными условиями, связанными с патентами.
4. **BSD License**: Семейство лицензий с минимальными ограничениями на использование и распространение.
5. **Creative Commons**: Различные лицензии для контента, которые могут применяться к репозиториям.
6. **Mozilla Public License 2.0 (MPL)**: Лицензия с требованиями относительно исходного кода, но с большей гибкостью по сравнению с GPL.
7. **Unlicense**: Публичное доменное разрешение на использование кода без ограничений.

При создании репозитория на GitHub можно выбрать одну из предлагаемых лицензий или добавить собственную лицензию, предоставив текст лицензии в репозитории.

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

**Клонирование репозитория GitHub** осуществляется с помощью команды `git clone` в терминале. Для этого необходимо скопировать URL репозитория (HTTPS или SSH) и выполнить команду:

```
git clone https://github.com/username/repository.git
```

```
git clone git@github.com:username/repository.git
```

**Зачем нужно клонировать репозиторий**:

1. **Локальная работа**: Получить полную копию репозитория на локальную машину для разработки, внесения изменений и тестирования.
2. **Вклад в проект**: Участвовать в развитии проекта, внося изменения и отправляя их обратно через пулл-запросы.
3. **Изучение кода**: Анализировать исходный код проекта для обучения или понимания его работы.
4. **Резервное копирование**: Иметь локальную копию репозитория как резервную копию данных.

### 14. Как проверить состояние локального репозитория Git?

Для проверки состояния локального репозитория Git используется команда:

```
git status
```

Эта команда отображает:

- Текущую ветку.
- Измененные файлы, которые еще не добавлены в индекс.
- Файлы, подготовленные для коммита.
- Неотслеживаемые файлы.
- Информацию о различиях между рабочим каталогом и индексом или последним коммитом.

Также можно использовать другие команды для получения дополнительной информации:

- `git log` — просмотреть историю коммитов.
- `git diff` — увидеть различия между рабочим каталогом и индексом.
- `git diff --staged` — увидеть различия между индексом и последним коммитом.

### 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?

**Последовательность операций и изменения состояний:**

1. **Добавление/изменение файла в рабочем каталоге**:
    - **Состояние файла**: `Untracked` (новый файл) или `Modified` (измененный файл).
    - `git status` показывает измененные или неотслеживаемые файлы.
2. **Добавление файла под версионный контроль с помощью `git add`**:
    - **Состояние файла**: `Staged`.
    - Файл готов к коммиту.
    - `git status` показывает файл в индексе для следующего коммита.
3. **Фиксация изменений с помощью `git commit`**:
    - **Состояние файла**: `Committed`.
    - Изменения сохраняются в локальной истории репозитория.
    - Рабочий каталог и индекс обновляются до состояния последнего коммита.
    - `git status` показывает, что рабочий каталог чист.
4. **Отправка изменений на сервер с помощью `git push`**:
    - **Состояние локального репозитория**: синхронизирован с удаленным репозиторием.
    - Удаленный репозиторий обновляется с новыми коммитами.
    - Локальный репозиторий остается в состоянии `Committed`, готовым к дальнейшим изменениям.

**Итоговая последовательность состояний:** `Untracked/Modified` → `Staged` → `Committed` → `Push` (синхронизация с удаленным репозиторием).

**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.**

**Последовательность команд для синхронизации двух локальных репозиториев с удаленным репозиторием на GitHub:**

**На первом компьютере (Computer A):**

**Клонирование репозитория:**
```
git clone https://github.com/username/repository.git
cd repository
```

**Внесение изменений:**

**Добавление изменений в индекс:**
```
git add .
```

**Создание коммита:**
```
git commit -m "Описание изменений"
```

**Отправка изменений на GitHub:**
```
git push origin main
```

**На втором компьютере (Computer B):**
**Клонирование репозитория (если еще не сделано):**

```
git clone https://github.com/username/repository.git
cd repository
```

**Получение последних изменений с GitHub:**
```
git pull origin main
```

**Внесение изменений:**

- Добавьте или измените файлы в рабочем каталоге.

**Добавление изменений в индекс:**
```
git add .
```

**Создание коммита:**
```
git commit -m "Описание изменений"
```

**Отправка изменений на GitHub:**
```
git push origin main
```

**Возвращение к Computer A:**

**Получение последних изменений с GitHub:**
```
git pull origin main
```

**Общая последовательность для синхронизации:**

- **Перед началом работы** на любом компьютере всегда выполняйте `git pull origin main` для получения последних изменений.
- **После внесения и коммита изменений** выполняйте `git push origin main` для отправки их на GitHub.
- Это гарантирует, что оба локальных репозитория остаются синхронизированными с удаленным репозиторием и между собой.

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

**Другие популярные сервисы, работающие с Git:**

1. **GitLab**: Предоставляет хостинг репозиториев, CI/CD, управление проектами и другими инструментами для разработки.
2. **Bitbucket**: От Atlassian, интегрируется с другими инструментами компании, такими как Jira.
3. **SourceForge**: Один из старейших хостингов для открытых проектов.
4. **Azure Repos**: Часть Azure DevOps от Microsoft, предлагает хостинг Git репозиториев.
5. **Gitea**: Легковесная само-хостинговая платформа для Git репозиториев.
6. **AWS CodeCommit**: Сервис от Amazon для хостинга Git репозиториев.

**Сравнительный анализ GitLab с GitHub:**

**1. Функциональность:**

- **GitHub**:
  - Основной фокус на хостинге Git репозиториев, сотрудничестве и сообществе.

    - Предоставляет инструменты для управления проектами, Issues, Pull Requests.
    - GitHub Actions для CI/CD.
    - Большое сообщество и поддержка открытых проектов.
- **GitLab**:
    - Более интегрированная платформа, включающая полный цикл DevOps: от планирования до мониторинга.
    - Встроенные инструменты для CI/CD, автоматизации, безопасности.
    - Поддержка само-хостинга и облачного хостинга.
    - Возможность более тонкой настройки процессов разработки.

**2. Модель хостинга:**

- **GitHub**:
    - В основном облачный сервис, предлагает также GitHub Enterprise для корпоративного использования.
    - Не поддерживает полнофункциональное само-хостинг (только GitHub Enterprise Server).
- **GitLab**:
    - Предоставляет как облачную, так и само-хостируемую версии.
    - Более гибкие возможности для настройки и интеграции в корпоративные инфраструктуры.

**3. Ценообразование:**

- **GitHub**:
    - Бесплатные публичные репозитории, ограниченное количество приватных репозиториев для бесплатных аккаунтов.
    - Платные планы для дополнительных функций и приватных репозиториев.
- **GitLab**:
    - Бесплатные и платные планы с различными уровнями функциональности.
    - Бесплатные приватные репозитории и более расширенные возможности в платных планах.

**4. Интеграции:**

- **GitHub**:
    - Широкая экосистема интеграций через GitHub Marketplace.
    - Хорошо интегрируется с другими инструментами разработки.
- **GitLab**:
    - Встроенные инструменты для многих этапов разработки.
    - Также поддерживает интеграции с внешними сервисами.

**5. Пользовательский интерфейс:**

- **GitHub**:
    - Интуитивно понятный и простой интерфейс, удобный для новичков и профессионалов.
- **GitLab**:
    - Более сложный интерфейс из-за обширной функциональности, что может требовать некоторого времени для освоения.

**6. Открытость:**

- **GitHub**:
    - Закрытая платформа, хотя GitHub открывает некоторые API и инструменты для разработчиков.
- **GitLab**:
    - Является открытым исходным кодом, что позволяет пользователям вносить изменения и адаптировать платформу под свои нужды.

**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.**

**Известные программные средства с графическим интерфейсом пользователя (GUI) для работы с Git:**

1. **GitHub Desktop**: Официальный клиент от GitHub для управления репозиториями, коммитов и пулл-запросов.
2. **Sourcetree**: Бесплатный клиент от Atlassian для управления Git и Mercurial репозиториями.
3. **GitKraken**: Популярный кросс-платформенный клиент с интуитивным интерфейсом.
4. **TortoiseGit**: Расширение для Windows, интегрированное с проводником.
5. **SmartGit**: Клиент с поддержкой Git, SVN и Mercurial.
6. **Visual Studio Code**: Редактор с встроенной поддержкой Git и множеством расширений.
7. **Fork**: Легковесный и быстрый Git клиент для Windows и macOS.

**Реализация операций Git с помощью GitHub Desktop:**

Предположим, что в лабораторной работе рассматриваются такие операции как клонирование репозитория, внесение изменений, добавление файлов, коммит, пуш и пулл.

1. **Клонирование репозитория:**
   - Откройте GitHub Desktop.
   - Нажмите на "File" → "Clone repository".
   - Выберите нужный репозиторий из списка или введите URL и нажмите "Clone".
2. **Внесение изменений:**
   - Откройте проект в предпочитаемом редакторе.
   - Внесите необходимые изменения или добавьте новые файлы.
3. **Добавление файлов и коммит:**
   - Вернитесь в GitHub Desktop. Измененные файлы будут отображены во вкладке "Changes".
   - Введите сообщение коммита в поле "Summary" (и при необходимости "Description").
   - Отметьте файлы для коммита (обычно все измененные файлы).
   - Нажмите кнопку "Commit to main" (или другой текущей ветки).
4. **Отправка изменений на сервер (Push):**
   - После коммита появится кнопка "Push origin". Нажмите ее, чтобы отправить изменения на GitHub.
5. **Получение изменений с сервера (Pull):**
   - Нажмите на кнопку "Fetch origin" или "Pull origin", чтобы получить и интегрировать изменения из удаленного репозитория.
6. **Создание новой ветки:**
   - Нажмите на текущую ветку в верхней части интерфейса и выберите "New Branch".
   - Введите имя ветки и нажмите "Create Branch".
7. **Слияние веток:**
   - Переключитесь на ветку, в которую нужно слить изменения.
   - Выберите "Branch" → "Merge into current branch".
   - Выберите ветку для слияния и подтвердите операцию.