

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:
Горбунов Данила Евгеньевич
2 курс, группа ИВТ-б-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 г.

Тема: Исследование основных возможностей Git и GitHub.

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

Теоретические сведения

Системы контроля версий

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.

Централизованные системы контроля версий

Следующая серьёзная проблема, с которой сталкиваются люди, — это необходимость взаимодействовать с другими разработчиками. Для того, чтобы разобраться с ней, были разработаны централизованные системы контроля версий (ЦСКВ). Такие системы, как CVS, Subversion и Perforce, используют единственный сервер, содержащий все версии файлов, и некоторое количество клиентов, которые получают файлы из этого централизованного хранилища. Применение ЦСКВ являлось стандартом на протяжении многих лет.

Целостность Git

В Git для всего вычисляется хеш-сумма, и только потом происходит сохранение. В дальнейшем обращение к сохранённым объектам происходит по этой хеш-сумме. Это значит, что невозможно изменить содержимое файла или директории так, чтобы Git не узнал об этом. Данная функциональность встроена в Git на низком уровне и является неотъемлемой частью его философии. Вы не потеряете информацию во время её передачи и не получите повреждённый файл без ведома Git.

Механизм, которым пользуется Git при вычислении хеш-сумм, называется SHA-1 хеш. Это строка длиной в 40 шестнадцатеричных

символов (0-9 и a-f), она вычисляется на основе содержимого файла или структуры каталога.

Выполнение работы

1. Зарегистрировался на GitHub и создал новый репозиторий. (Рисунок 1)

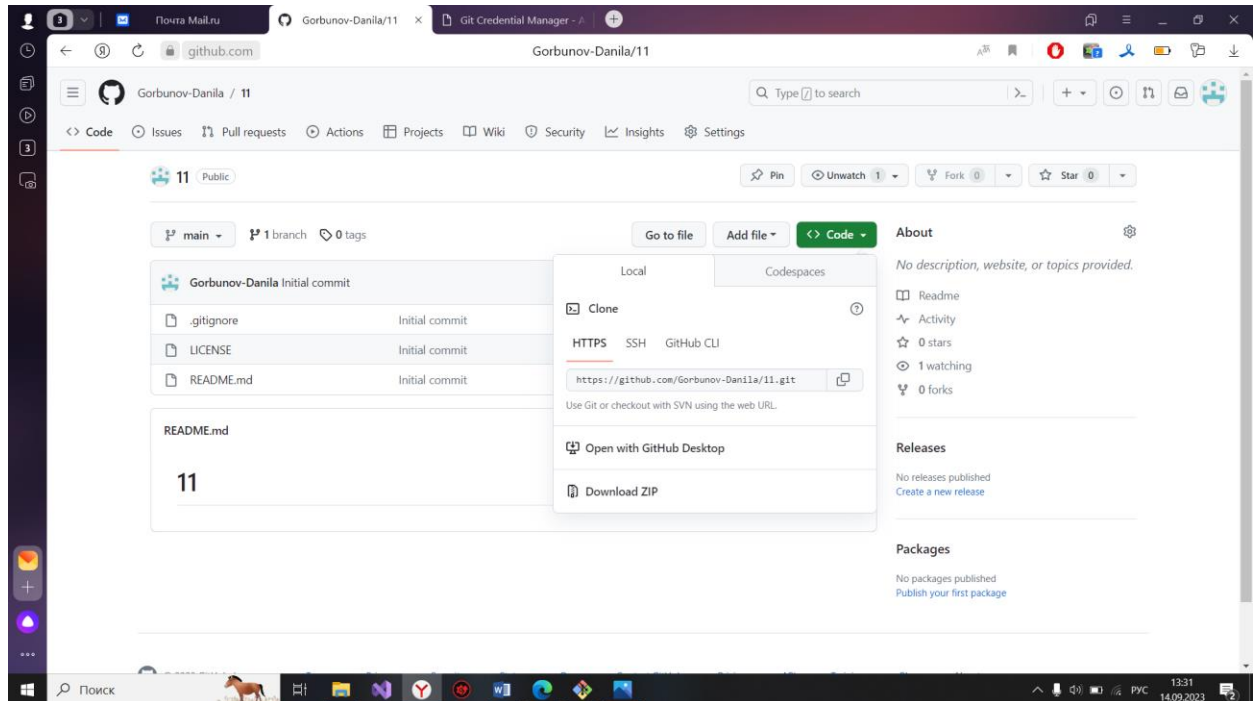



Рисунок 1. Репозиторий

2. Скопировал репозиторий в Git. (Рисунок 2)

```
Admin@DESKTOP-TSLUNFU MINGW64 ~ (main)
$ git clone https://github.com/Gorbunov-Danila/11.git
Cloning into '11'...
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.
```

Рисунок 2. Копирование

3. Дополнил файл .gitignore необходимыми правилами для выбранного языка программирования. (Рисунок 3)

 .gitignore – Блокнот
Файл Правка Формат Вид Справка

```
# 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
```

Рисунок 3. .gitignore

4. Локально изменил содержимое. (Рисунок 4)

```

Admin@DESKTOP-TSLUNFU MINGW64 ~ (main)
$ cd C:\\Users\\Admin\\11

Admin@DESKTOP-TSLUNFU MINGW64 ~/11 (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean

Admin@DESKTOP-TSLUNFU MINGW64 ~/11 (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")

```

Рисунок 4. Содержимое

5. Распространил изменения в исходный репозиторий GitHub. (Рисунок 5)

```

Admin@DESKTOP-TSLUNFU MINGW64 ~/11 (main)
$ git pull
Already up to date.

Admin@DESKTOP-TSLUNFU MINGW64 ~/11 (main)
$ git remote add origin https://github.com/Gorbunov-Danila/11.git
error: remote origin already exists.

Admin@DESKTOP-TSLUNFU MINGW64 ~/11 (main)
$ git push --set upstream origin main
fatal: 'upstream' does not appear to be a git repository
fatal: Could not read from remote repository.

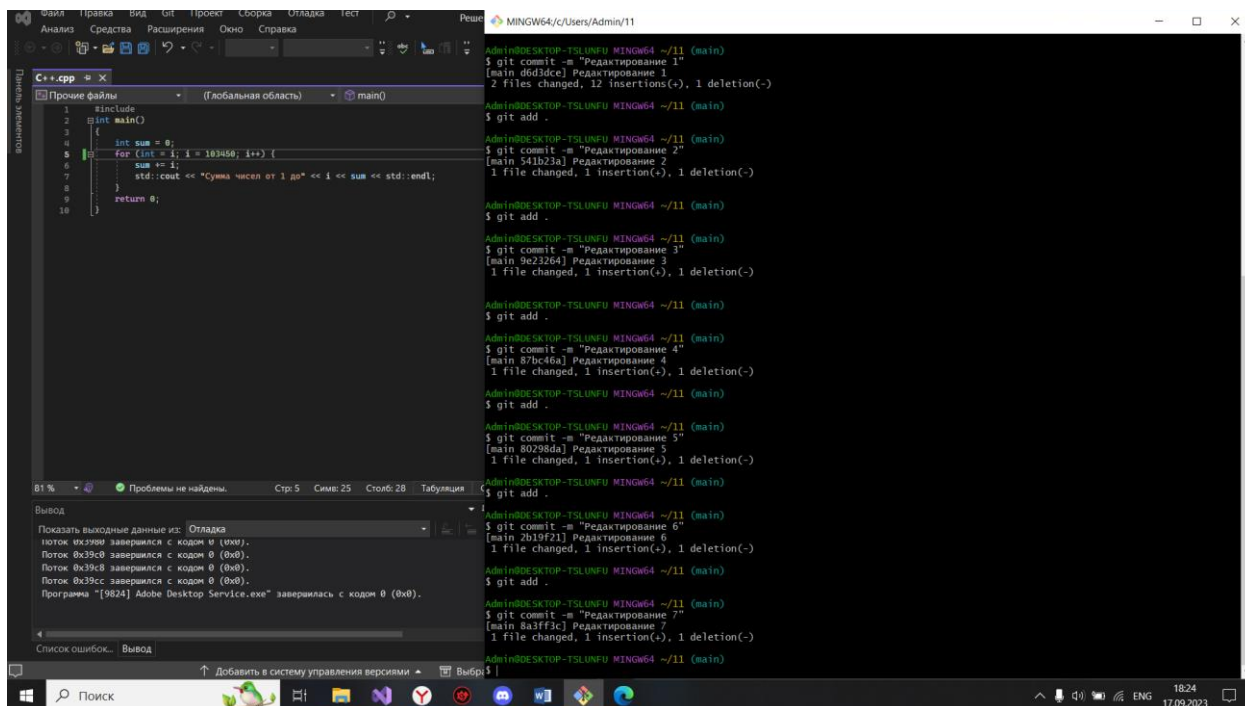
Please make sure you have the correct access rights
and the repository exists.

Admin@DESKTOP-TSLUNFU MINGW64 ~/11 (main)
$ git push --set-upstream origin main
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), 256 bytes | 256.00 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/Gorbunov-Danila/11.git
   6ab8fe8..3e2d8ae  main -> main
branch 'main' set up to track 'origin/main'.

```

Рисунок 5. Распространение репозитория

6. Написал программу и выполнил 7 коммитов. (Рисунок 6)



The screenshot displays a Windows desktop environment. On the left, the Visual Studio IDE is open, showing a C++ program in a file named 'main.cpp'. The code is as follows:

```
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     int sum = 0;
7     for (int i = 1; i <= 10; i++) {
8         sum += i;
9     }
10    cout << "Сумма чисел от 1 до 10: " << sum << endl;
11    return 0;
12 }
```

On the right, a terminal window titled 'MINGW64/c/Users/Admin/11' shows a series of 7 git commits. Each commit is preceded by a prompt 'Admin@DESKTOP-TSLUNFU MINGW64 ~/11 (main)'. The commits are as follows:

- Commit 1: \$ git commit -m "Редактирование 1" [main d6d3dce] Редактирование 1 2 files changed, 12 insertions(+), 1 deletion(-)
- Commit 2: \$ git add . \$ git commit -m "Редактирование 2" [main 541b23a] Редактирование 2 1 file changed, 1 insertion(+), 1 deletion(-)
- Commit 3: \$ git add . \$ git commit -m "Редактирование 3" [main 9e2226a] Редактирование 3 1 file changed, 1 insertion(+), 1 deletion(-)
- Commit 4: \$ git add . \$ git commit -m "Редактирование 4" [main 87bc46a] Редактирование 4 1 file changed, 1 insertion(+), 1 deletion(-)
- Commit 5: \$ git add . \$ git commit -m "Редактирование 5" [main 80298da] Редактирование 5 1 file changed, 1 insertion(+), 1 deletion(-)
- Commit 6: \$ git add . \$ git commit -m "Редактирование 6" [main 2b19f21] Редактирование 6 1 file changed, 1 insertion(+), 1 deletion(-)
- Commit 7: \$ git add . \$ git commit -m "Редактирование 7" [main 6a3ff3c] Редактирование 7 1 file changed, 1 insertion(+), 1 deletion(-)

The bottom of the terminal window shows the prompt 'Admin@DESKTOP-TSLUNFU MINGW64 ~/11 (main) \$'.

Рисунок 6. Программа и 7 коммитов

Контрольные вопросы

1. СКВ - это аббревиатура, которая означает "Система Контроля Версий". Это программное обеспечение, предназначенное для управления изменениями в исходном коде, документах и других файловых системах в проекте. СКВ играет важную роль в разработке программного обеспечения и управлении проектами.

2. Недостатки локальных СКВ:

Отсутствие центрального сервера: Локальные СКВ не предлагают единого центрального сервера для хранения репозитория. Это может сделать сложным обмен изменениями между разработчиками.

Ограниченное удаленное сотрудничество: Поскольку каждый разработчик работает со своей копией репозитория, передача изменений другим участникам проекта требует дополнительных шагов, например, отправки патчей или объединения файлов вручную.

Риски потери данных: Если устройство или хранение локальной копии репозитория теряется или повреждается без наличия альтернативной копии, это может привести к потере всех версий данных и истории изменений.

Недостатки централизованных СКВ:

Единственная точка отказа: В централизованных СКВ все данные хранятся на центральном сервере. Если сервер недоступен или происходит сбой, разработчики не могут получить доступ к репозиторию и продолжать работу.

Зависимость от скорости сети: Так как каждый запрос на коммит или обновление требует связи с центральным сервером, эффективность работы в значительной степени зависит от скорости соединения.

Ограничение контроля версий: Централизованные СКВ могут предоставлять ограниченные возможности для работы с распределенными командами и ветвлением (branching), что может затруднить параллельную разработку и управление сложными проектами.

Отсутствие локальных коммитов: В централизованных СКВ все изменения отправляются на центральный сервер, поэтому нет возможности делать локальные коммиты без доступа к серверу.

3. Git относится к распределенным системам контроля версий (СКВ). В отличие от централизованных СКВ, где все данные хранятся на центральном сервере, Git предоставляет каждому разработчику полную локальную копию репозитория. Это означает, что каждый разработчик может работать независимо и иметь всю историю изменений на своем компьютере.

4. Распределенная модель: Одним из ключевых отличий Git является его распределенная модель. Каждый разработчик имеет полную локальную копию репозитория, включая всю историю изменений. Это позволяет разработчикам работать независимо без необходимости постоянного подключения к центральному серверу. Распределенная модель Git обеспечивает высокую гибкость и быстродействие.

Скорость и эффективность: Git разработан с учетом скорости и эффективности. Он использует механизмы сжатия данных, оптимизированные алгоритмы для работы с изменениями и эффективную обработку больших репозиториях. Это делает Git очень быстрым и масштабируемым в сравнении с другими СКВ.

5. Целостность хранимых данных в Git обеспечивается с помощью использования уникальных хеш-кодов для каждого коммита и контрольной суммы (SHA-1) для всех файлов. Каждый коммит имеет свой уникальный идентификатор, который зависит от содержимого коммита. При изменении или повреждении данных, таких как файлов или метаданных репозитория, будет изменяться и уникальный хеш-код.

6. В Git файлы могут находиться в трех основных состояниях:

Измененные (Modified): Это состояние, когда файлы были изменены, но изменения еще не были зафиксированы в коммите. Git отслеживает изменения в файлах и помечает их как измененные.

В индексе (Staged): Когда измененные файлы добавляются в индекс, они переходят в состояние "в индексе". Индекс - это промежуточная область, где Git готовит файлы к коммиту. Файлы, находящиеся в индексе, уже отмечены для включения в следующий коммит.

Зафиксированные (Committed): Когда файлы зафиксированы в коммите, они считаются зафиксированными. В этом состоянии файлы сохранены в локальном репозитории. Это означает, что изменения в файлах сохранены и могут быть восстановлены в любое время.

7. Профиль пользователя в GitHub представляет собой публичную страницу, где пользователь может представить себя, свои проекты и внести информацию о своей деятельности на платформе GitHub. Профиль пользователя является основным местом для отображения и подведения итогов его работы и вклада в сообщество разработчиков.

8. 1) Публичные репозитории: доступные для общего просмотра и использования любым пользователем GitHub.

2) Частные репозитории: доступны только для авторизованных пользователей и могут быть использованы для сохранения приватной информации или коммерческих проектов.

3) Форкнутые репозитории: копии существующих репозиториях, созданные пользователями для модификации и внесения своих изменений без воздействия на исходный репозиторий.

4) Шаблоны репозиториях: предварительно созданные шаблоны, которые служат основой для проекта, позволяя пользователям быстро начать работу над новым проектом.

5) Организационные репозитории: репозитории, принадлежащие организации, а не индивидуальному пользователю.

6) Архивные репозитории: репозитории, которые больше не активно разрабатываются или поддерживаются, но по-прежнему доступны для просмотра и использования.

9. 1) Создание репозитория: Пользователь создает новый репозиторий на GitHub, назначая ему имя и описание.

2) Клонирование репозитория: Пользователь клонирует репозиторий на свой компьютер с помощью команды `git clone`.

3) Добавление изменений: Пользователь вносит изменения в файлы в локальной копии репозитория.

4) Отправка изменений: Пользователь добавляет изменения в комит с помощью команды `git add` и фиксирует комит с помощью команды `git commit`.

5) Пуш изменений: Пользователь отправляет комит на удаленный репозиторий с помощью команды `git push`.

6) Работа с ветками: Пользователь может создавать и переключаться между ветками с помощью команд `git branch` и `git checkout`.

7) Пулл-реквесты: Пользователь может создавать пулл-реквесты, чтобы предлагать изменения в основную ветку репозитория.

8) Ревью кода: Пользователь может просматривать и комментировать пулл-реквесты других пользователей.

9) Слияние изменений: Пользователь может принять или отклонить пулл-реквесты и слить изменения в основную ветку репозитория.

10) Синхронизация с удаленным репозиторием: Пользователь может получать изменения из удаленного репозитория с помощью команды `git pull` и отправлять свои изменения с помощью команды `git push`.

10. 1. Установите Git на свой компьютер, если еще не сделали этого.

2. Откройте командную строку или терминал и выполните команду для настройки имени пользователя:

```
git config --global user.name «Ваше имя»
```

3. Затем настройте свою электронную почту:

```
git config --global user.email «ваша почта»
```

4. Настройте редактор, который будет использоваться Git при создании сообщений коммитов:

```
git config --global core.editor «название_редактора»
```

Например, для использования редактора Nano, выполните:

```
git config --global core.editor «nano»
```

5. Проверьте свои настройки, выполнив команду

```
git config --list
```

Она покажет ваши текущие настройки Git.

Теперь ваша основная настройка Git завершена, и вы готовы начать работу с Git на своем компьютере.

11. 1) Зарегистрируйтесь на GitHub, если у вас еще нет аккаунта. Вам потребуется электронный адрес и пароль для регистрации.

2) После успешной регистрации войдите в свой аккаунт на GitHub.

3) На главной странице GitHub, в верхнем правом углу, нажмите на значок «+», а затем выберите «New repository» (Новый репозиторий).

4) Введите имя для своего репозитория. Обычно рекомендуется использовать осмысленное имя, связанное с проектом, чтобы другим пользователям было легче понять, что содержится в репозитории.

5) Добавьте краткое описание к своему репозиторию. Описание может включать информацию о цели или назначении проекта.

6) Выберите тип репозитория: публичный (public) или приватный (private). Публичный репозиторий виден всем пользователям GitHub, а приватный — только вам и тем, кого вы укажете.

7) Укажите файлы, которые будут автоматически добавлены в репозиторий при создании (например, лицензию или README.md).

8) Нажмите на кнопку «Create repository» (Создать репозиторий), чтобы завершить процесс создания репозитория.

12. На GitHub поддерживается несколько типов лицензий при создании репозитория, включая:

1) MIT License: открытая лицензия, позволяющая свободное использование, модификацию и распространение кода. Часто используется в проектах с открытым исходным кодом.

2) GNU General Public License (GPL): открытая лицензия, которая гарантирует свободу использования, изменения и распространения проекта, при условии, что все производные работы также будут открыты.

3) Apache License: открытая лицензия, предназначенная для работы с программным обеспечением открытого исходного кода. Позволяет использовать, модифицировать и распространять код, как с открытым, так и с закрытым исходным кодом.

4) Creative Commons License: не является лицензией для программного обеспечения, но используется для установления правил использования, распространения и модификации других типов контента, таких как тексты, изображения или музыка.

13. Клонирование репозитория GitHub осуществляется с помощью команды ``git clone`` в командной строке. Для клонирования нужно указать URL репозитория.

Зачем нужно клонировать репозиторий:

1) Работа над проектом: Клонирование репозитория позволяет получить локальную копию проекта, над которым можно работать независимо от других разработчиков. Вы можете вносить изменения, создавать ветки и просматривать историю изменений без влияния на оригинальный репозиторий.

2) Доступ к исходному коду: Клонирование репозитория позволяет получить исходный код проекта и изучать его. Это полезно, например, для изучения работы определенных функций, алгоритмов или понимания архитектуры проекта.

3) Сотрудничество и публичные запросы на внесение изменений: Клонирование репозитория необходимо, если вы хотите сотрудничать над проектом или создать публичный запрос на внесение изменений (pull request). Клонирование репозитория помогает вам создать локальную копию, над которой вы будете работать, прежде чем внести изменения в оригинальный репозиторий.

14. Чтобы проверить состояние локального репозитория Git, выполните команду «git status». Эта команда покажет информацию о текущей ветке, изменениях в рабочей директории и индексе. Она покажет, какие файлы были изменены, добавлены или удалены, и они находятся в состоянии «отслеживаемых» или «неотслеживаемых». Также будет информация о том, есть ли изменения, которые не были зафиксированы коммитом, и если есть, то это будет отображено в разделе «Changes not staged for commit».

15. После выполнения каждой из этих операций состояние локального репозитория Git будет изменяться следующим образом:

1) Добавление/изменение файла в локальный репозиторий Git:

- Файл будет добавлен в рабочую директорию Git.
- Файл будет отображен как измененный в разделе «Changes not staged for commit» при выполнении команды git status.

2) Добавление нового/измененного файла под версионный контроль с помощью команды `git add`:

- Файл будет добавлен в индекс (staging area) Git для фиксации.
- Файл будет отображен как измененный в разделе «Changes to be committed» при выполнении команды `git status`.

3) Фиксация (коммит) изменений с помощью команды `git commit`:

- Изменения в индексе (staging area) будут зафиксированы.
- Коммит будет создан с указанными изменениями и фиксирующим сообщением.
- Коммит будет добавлен в историю изменений репозитория Git.

4) Отправка изменений на сервер с помощью команды `git push`:

- Все локальные коммиты, которые еще не были отправлены на удаленный сервер, будут отправлены.
- Изменения будут включены в удаленную ветку репозитория Git.

16. Для обновления и синхронизации локальных репозитория с репозиторием на GitHub, можно использовать следующую последовательность команд:

1) На каждом рабочем компьютере, откройте командную строку или терминал.

2) Введите команду `'git clone <ссылка_на_репозиторий_GitHub>'` для создания локальной копии репозитория с GitHub. Замените `'<ссылка_на_репозиторий_GitHub>'` на фактическую ссылку на Ваш репозиторий на GitHub.

3) Перейдите в каталог репозитория с помощью команды `'cd <название_репозитория>'`. Замените `'<название_репозитория>'` на фактическое название Вашего репозитория.

4) Убедитесь, что Вы находитесь на главной (master) ветке репозитория, выполнив команду `'git checkout master'`. В случае, если Вы работаете с другой веткой, повторите эту команду, заменив `'master'` на название нужной ветки.

5) Для получения последних изменений из репозитория на GitHub, выполните команду ``git pull origin master``. Здесь ``origin`` - это название удаленного репозитория на GitHub, а ``master`` - это название ветки на удаленном репозитории.

6) Если Вы внесли изменения в код или файлы, и хотите загрузить их в репозиторий на GitHub, сначала добавьте файлы для коммита с помощью команды ``git add .``, где ``.`` означает все файлы.

7) Затем примените коммит с описанием изменений, используя команду ``git commit -m "сообщение коммита"``. Замените ``"сообщение коммита"`` на описание Ваших изменений.

8) Загрузите коммит в репозиторий на GitHub с помощью команды ``git push origin master``. Это отправит Ваши локальные изменения на удаленный репозиторий на GitHub.

После выполнения этих команд, оба локальных репозитория будут находиться в синхронизированном состоянии с репозиторием на GitHub.

17. Некоторые альтернативные сервисы, работающие с Git, включают GitLab, Bitbucket и Azure DevOps.

Один из наиболее известных альтернативных сервисов Git - GitLab. GitLab предлагает множество функций, подобных GitHub, но имеет несколько отличий.

1) Форма развертывания: GitLab является как облачным сервисом, так и самостоятельным приложением для самостоятельного развертывания на собственном сервере. GitHub основан на облачных ресурсах.

2) Модель лицензирования: GitLab имеет открытое и закрытое ПО. Основная функциональность, включая публичные репозитории, доступна бесплатно, но есть также платные планы для предоставления дополнительных возможностей. В отличие от этого, GitHub был предназначен только для работы с открытыми репозиториями на протяжении долгого времени, хотя в настоящее время предлагает некоторые функции для частных репозиториях.

3) Возможности CI/CD: GitLab интегрирует непрерывную интеграцию и непрерывное развертывание (CI/CD) непосредственно в свою платформу, предлагая встроенную поддержку автоматического выполнения тестов и развертывания при изменении в репозитории. Тогда как GitHub предлагает интеграцию с популярными инструментами CI/CD, такими как Travis CI и Jenkins, но не имеет встроенных инструментов для CI/CD.

4) Управление задачами: GitLab предоставляет возможности управления задачами (issue tracking) в своей платформе, позволяя пользователям создавать задачи, отслеживать их выполнение и комментировать. GitHub также предлагает управление задачами, но его функциональность более ограничена.

5) Коллаборация: Оба сервиса предлагают возможность для совместной работы над проектами. Однако GitLab предлагает более широкий набор возможностей для коллаборации, таких как совместное редактирование кода в реальном времени, встроенный code review и комментарии на уровне строки кода.

Итак, самыми существенными отличиями GitLab от GitHub являются его форма развертывания (возможность самостоятельного развертывания на собственном сервере), модель лицензирования (открытое и закрытое ПО), интеграция с CI/CD (встроенная поддержка) и возможности управления задачами (issue tracking). Оба сервиса поддерживают открытые и приватные репозитории и предлагают возможности для совместной работы над проектами. В конечном счете, выбор между GitLab и GitHub зависит от индивидуальных потребностей и предпочтений разработчиков.

18. Некоторые известные программные средства с графическим интерфейсом для работы с Git включают в себя:

GitHub Desktop: Позволяет клонировать репозитории, создавать и переключаться между ветками, фиксировать изменения, выполнять пуш и пулл, а также просматривать историю коммитов.

GitKraken: Обладает многофункциональным интерфейсом для управления репозиториями, включая создание веток, фиксацию изменений, решение конфликтов, просмотр истории и другие операции.

Sourcetree: Предоставляет графический интерфейс для клонирования, фиксации изменений, визуализации веток и слияния.

GitExtensions: Обладает широким набором функций, включая визуализацию истории, управление ветками, статистику и редактирование .gitignore.

TortoiseGit: Этот инструмент интегрируется в проводник Windows и предоставляет контекстное меню для выполнения действий Git, таких как клонирование, фиксация изменений и слияние.

Пример реализации операций Git с помощью GitHub Desktop:

Запустите GitHub Desktop.

Выберите "File" > "Clone Repository" (Клонировать репозиторий) для клонирования.

Во вкладке "Current Repository" (Текущий репозиторий) выберите репозиторий.

Выберите "Branch" (Ветка) для создания новой ветки.

Внесите изменения в файлы и фиксируйте их, выбрав "Commit" (Фиксировать).

Отправьте изменения, выбрав "Push" (Отправить).

Обратите внимание, что операции могут немного различаться в зависимости от используемой программы.

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