

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ПРАКТИЧЕСКОЙ РАБОТЕ №1
дисциплины «Программирование на Python»

Выполнила:
Коробка В.А.,
2 курс, группа ИВТ-б-о-24-1,
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., доцент департамента
цифровых, робототехнических систем
и электроники института
перспективной инженерии

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2025 г.

Тема: Исследование основных возможностей Git и GitHub.

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

Адрес репозитория: <https://github.com/13r4lera/my-project>.

Конспект теоретического материала

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.

Многие люди в качестве метода контроля версий применяют копирование файлов в отдельную директорию. Данный подход очень распространён из-за его простоты, однако он невероятно сильно подвержен появлению ошибок. Для того, чтобы решить эту проблему, программисты разработали локальные СКВ с простой базой данных, которая хранит записи обо всех изменениях в файлах, осуществляя тем самым контроль ревизий. Одной из популярных СКВ была система RCS.

Следующая проблема - это необходимость взаимодействовать с другими разработчиками. Были разработаны централизованные системы контроля версий (ЦСКВ). Такие системы, как CVS, Subversion и Perforce, используют единственный сервер, содержащий все версии файлов. Самый очевидный минус — это единая точка отказа, представленная централизованным сервером.

Здесь в игру вступают распределённые системы контроля версий (РСКВ). В РСКВ клиенты полностью копируют репозиторий. В этом случае, если один из серверов умрёт, любой клиентский репозиторий может быть скопирован на другой сервер для продолжения работы. Каждая копия репозитория является полным бэкапом всех данных. Многие РСКВ могут одновременно взаимодействовать с несколькими удалёнными репозиториями.

Основное отличие Git от любой другой СКВ - это подход к работе со своими данными. Большинство других систем хранят информацию в виде списка изменений в файлах. Эти системы представляют хранимую

информацию в виде набора файлов и изменений, сделанных в каждом файле, по времени. Подход Git к хранению данных больше похож на набор снимков миниатюрной файловой системы.

В Git для всего вычисляется хеш-сумма, и только потом происходит сохранение. Механизм, которым пользуется Git при вычислении хеш-сумм, называется SHA-1 хеш.

Когда вы производите какие-либо действия в Git, практически все из них только добавляют новые данные в базу Git. Очень сложно заставить систему удалить данные либо сделать что-то, что нельзя впоследствии отменить. Как и в любой другой СКВ, вы можете потерять или испортить свои изменения, пока они не зафиксированы.

У Git есть три основных состояния, в которых могут находиться ваши файлы. Зафиксированный (committed) значит, что файл уже сохранён в вашей локальной базе. К изменённым (modified) относятся файлы, которые поменялись, но ещё не были зафиксированы. Подготовленные (staged) файлы - это изменённые файлы, отмеченные для включения в следующий коммит. Мы подошли к трём основным секциям проекта Git и область подготовленных файлов (staging area). Git-директория (Git directory) - это то место, где Git хранит метаданные и базу объектов вашего проекта. Рабочая директория (working directory) является снимком версии проекта. Область подготовленных файлов - это файл, обычно располагающийся в вашей Git-директории, в нём содержится информация о том, какие изменения попадут в следующий коммит. Эту область ещё называют "индекс", однако называть её stage-область также общепринято. Базовый подход в работе с Git выглядит так:

1. Вы изменяете файлы в вашей рабочей директории.
2. Вы выборочно добавляете в индекс только те изменения, которые должны попасть в следующий коммит, добавляя тем самым снимки только этих изменений в область подготовленных файлов.
3. Когда вы делаете коммит, используются файлы из индекса как есть, и этот снимок сохраняется в вашу Git-директорию. Если определённая версия

файла есть в Git-директории, эта версия считается зафиксированной. Если версия файла изменена и добавлена в индекс, значит, она подготовлена. И если файл был изменён с момента последнего распаковывания из репозитория, но не был добавлен в индекс, он считается изменённым.

GitHub – платформа для размещения кода. Во вкладке Code находятся два файла. README.md – файл, который описывает проект; каждый репозиторий должен включать этот файл. GitHub находит его и отображает его содержимое под репозиторием. Другой файл – gitignore – указывает, какие файлы и каталоги Git следует игнорировать. Репозиторий является общедоступным, но изменять файлы напрямую в нем можно, только являясь соавтором проекта. Можно сделать коммит или создать пулл-реквест, чтобы внести свой вклад в проект, даже если вы не являетесь соавтором. Вкладка Pull requests содержит пулл-реквесты – предложения об изменениях в файлах, находящихся в репозитории. Его владельцы могут рассмотреть запрос и внести ваши изменения, если сочтут их уместными. На вкладке Insight вы можете найти статистику и информацию о данном репозитории. Вы также можете посмотреть на коммиты, которые представляют изменения в содержании репозитория.

Порядок выполнения работы

На странице создания нового репозитория в Github заполнены поля Repository name, Description, Public. Поставлены галочки Add a README file, Add .gitignore (был выбран язык Python), Choose a license (выбрана MIT License). Создан репозиторий.

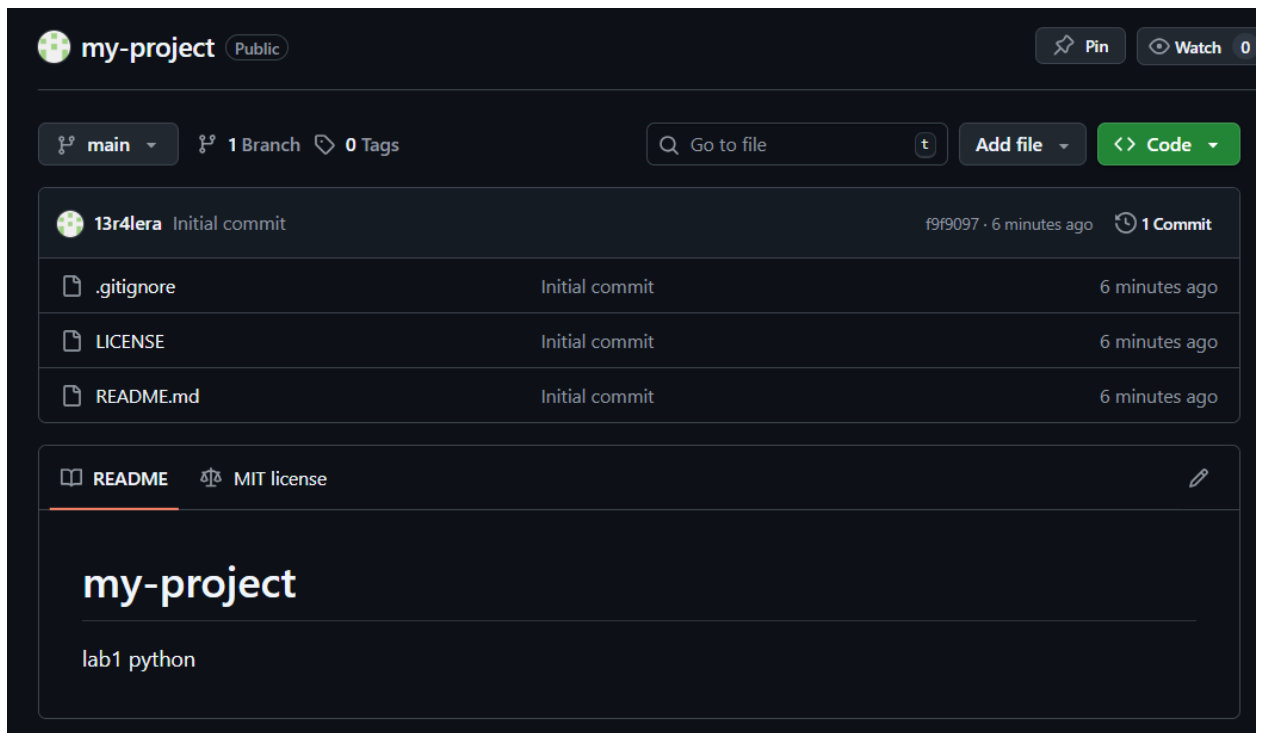


Рисунок 1. Репозиторий на Github

На рабочий компьютер был установлен Git. Был клонирован репозиторий на компьютер.

```
nout1@MSI MINGW64 ~
$ https://github.com/13r4lera/my-project.git
bash: https://github.com/13r4lera/my-project.git: No such file or directory

nout1@MSI MINGW64 ~
$ git clone https://github.com/13r4lera/my-project.git
Cloning into 'my-project'...
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 (from 0)
Receiving objects: 100% (5/5), done.

nout1@MSI MINGW64 ~
$ cd my-project

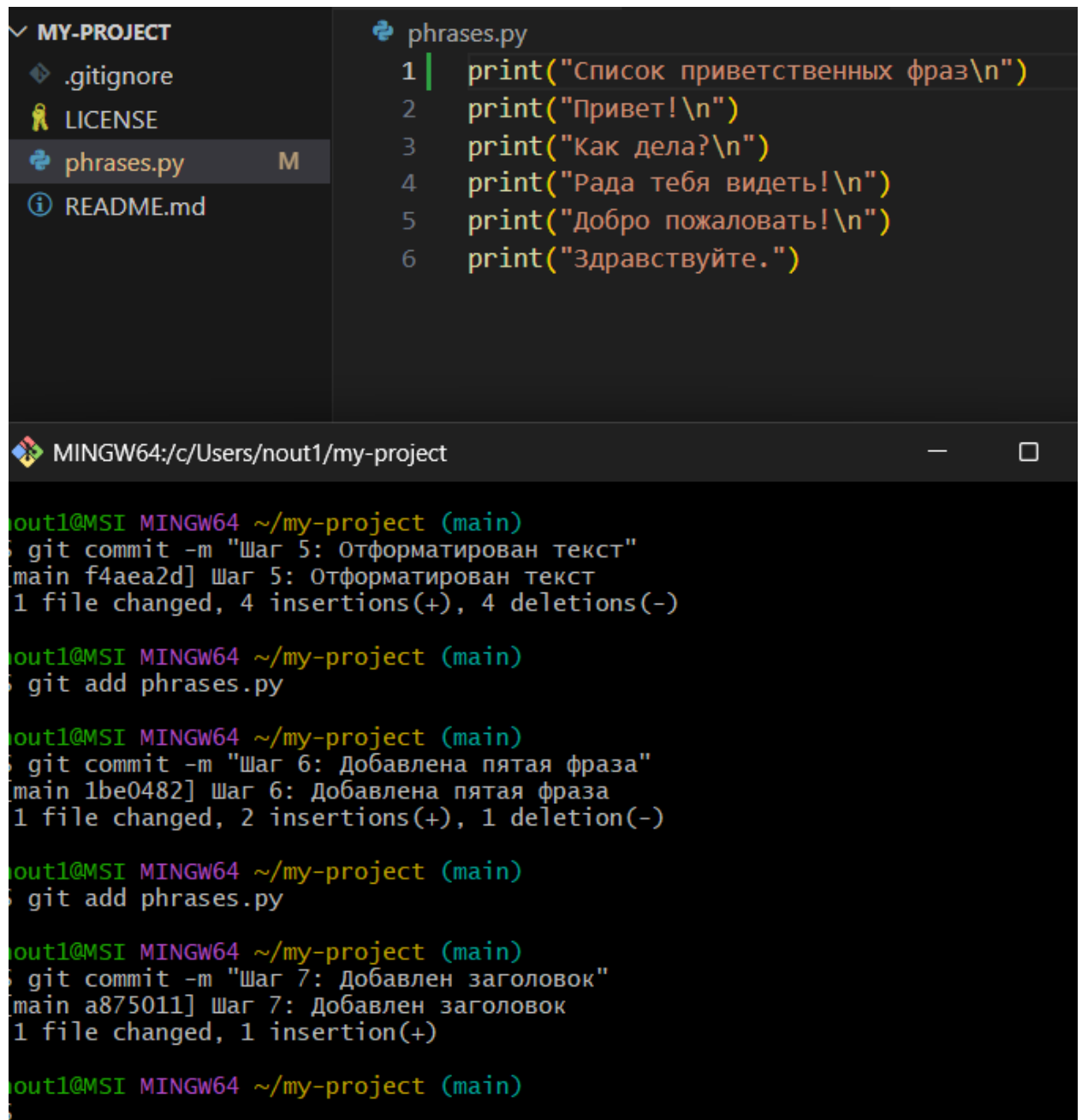
nout1@MSI MINGW64 ~/my-project (main)
$ |
```

Рисунок 2. Клонирование репозитория

В файл README.md добавлена информация о студенте, выполняющем лабораторную работу (ФИО и группа) и информация о лабораторной работе.

Был изменен файл .gitignore. В оригинальном стандартном .gitignore для Python содержатся правила для большого количества инструментов, библиотек и сценариев. Так как проект очень маленький и простой все эти правила были удалены, оставив только минимальный набор.

Написана небольшая программа на Python. При написании было зафиксировано 7 изменений в локальной репозитории, сделано 7 коммитов.



The screenshot shows a code editor window with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'MY-PROJECT' with files: .gitignore, LICENSE, phrases.py (marked with 'M' for modified), and README.md. The code editor shows the content of 'phrases.py' with six lines of Python code using the print function to output Russian phrases. Below the editor is a terminal window showing the execution of git commands to commit and add files, with output indicating the number of files changed and insertions/deletions.

```
MY-PROJECT
├── .gitignore
├── LICENSE
├── phrases.py M
└── README.md
```

```
phrases.py
1 | print("Список приветственных фраз\n")
2 | print("Привет!\n")
3 | print("Как дела?\n")
4 | print("Рада тебя видеть!\n")
5 | print("Добро пожаловать!\n")
6 | print("Здравствуйте.")
```

```
MINGW64:/c/Users/nout1/my-project

nout1@MSI MINGW64 ~/my-project (main)
$ git commit -m "Шаг 5: Отформатирован текст"
[main f4aea2d] Шаг 5: Отформатирован текст
1 file changed, 4 insertions(+), 4 deletions(-)

nout1@MSI MINGW64 ~/my-project (main)
$ git add phrases.py

nout1@MSI MINGW64 ~/my-project (main)
$ git commit -m "Шаг 6: Добавлена пятая фраза"
[main 1be0482] Шаг 6: Добавлена пятая фраза
1 file changed, 2 insertions(+), 1 deletion(-)

nout1@MSI MINGW64 ~/my-project (main)
$ git add phrases.py

nout1@MSI MINGW64 ~/my-project (main)
$ git commit -m "Шаг 7: Добавлен заголовок"
[main a875011] Шаг 7: Добавлен заголовок
1 file changed, 1 insertion(+)
```

Рисунок 3. Внесение изменений в программу.

После этого изменения в локальной репозитории были отправлены в удаленный репозиторий GitHub.

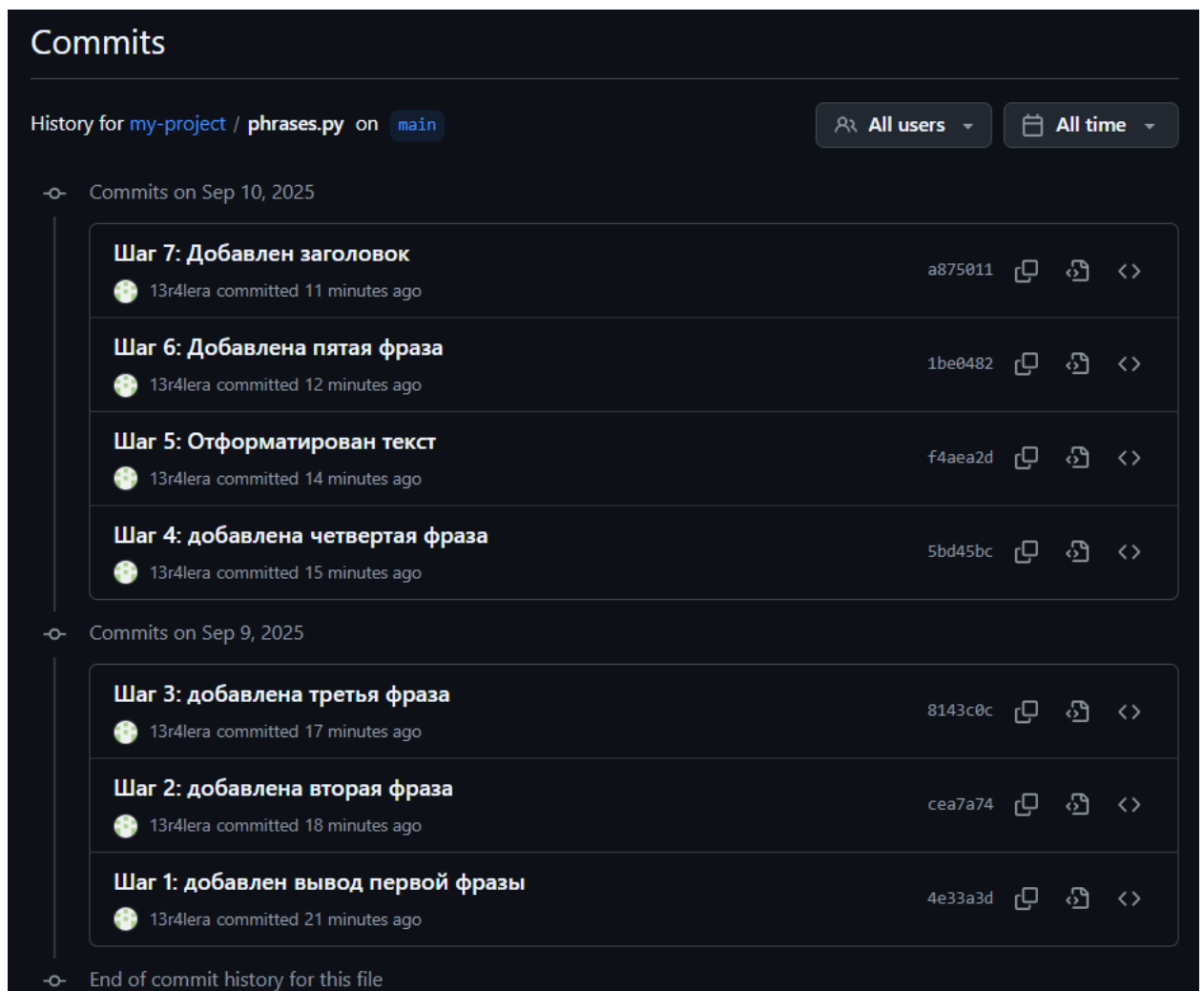


Рисунок 4. Коммиты в GitHub.

Ответы на контрольные вопросы

1. Что такое СКВ и каково ее назначение?

СКВ (Система контроля версий) - это инструмент для отслеживания изменений файлов и управления историей их версий. Она помогает сохранять историю изменений проекта, позволяет работать нескольким разработчикам одновременно, дает возможность отката к предыдущим версиям, позволяет управлять параллельной разработкой через ветки.

2. В чем недостатки локальных и централизованных СКВ?

В локальных СКВ все версии хранятся на одном компьютере, присутствует риск потери данных. Сложно работать команде, особенно распределенной. Централизованные СКВ требуют постоянное подключение к серверу. Если сервер недоступен, работа блокируется.

3. К какой СКВ относится Git?

Git — это распределенная СКВ.

4. В чем концептуальное отличие Git от других СКВ?

Git хранит полные снимки проекта, а не только изменения. Каждая копия репозитория является полноценной (не зависит от сервера).

5. Как обеспечивается целостность хранимых данных в Git?

Git использует SHA-1 хеширование для каждого коммита. Любое изменение в файле меняет хеш, что позволяет обнаружить повреждения.

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

Файл в Git может находиться в трёх состояниях: `modified` (изменён) - файл изменён, но не добавлен в индекс, `staged` (подготовлен к коммиту) — файл добавлен в индекс с помощью `git add`, `committed` (закоммичен) — файл сохранён в локальной истории. Связь: `Modified` - `Staged` - `Committed`.

7. Что такое профиль пользователя в GitHub?

Профиль GitHub — это личная страница пользователя, где хранится список репозиториев, информация о пользователе, активность и вклад в проекты.

8. Какие бывают репозитории в GitHub?

`Public` (публичные) - доступны всем пользователям. `Private` (приватные) - доступны только выбранным пользователям. `Internal` (внутренние) - доступны только членам организации (для корпоративных аккаунтов).

9. Укажите основные этапы модели работы с GitHub.

Создание репозитория. Клонирование репозитория на локальный компьютер. Внесение изменений в файлы. Добавление изменений в индекс (`git add`). Создание коммита (`git commit`). Отправка изменений на сервер (`git push`). Получение изменений от других (`git pull`).

10. Как осуществляется первоначальная настройка Git после установки?

```
git config --global user.name "Ваше имя"
```

```
git config --global user.email "ваш_email@example.com"
```

11. Опишите этапы создания репозитория в GitHub.

На GitHub нажать New repository. Задать имя, описание, тип репозитория (публичный/приватный). При необходимости добавить README, .gitignore, лицензию. Нажать Create repository.

12. Типы лицензий которые поддерживаются GitHub: MIT License, Apache License 2.0, GNU General Public License, BSD License, Creative Commons Zero v1.0 Universal, Mozilla Public License 2.0, The Unlicense, Academic Free License v3.0.

13. Чтобы клонировать репозиторий, нужно скопировать его URL, а затем в терминале с помощью команды `git clone` выполнить клонирование. Клонирование необходимо, чтобы: работать с кодом локально, использовать Git для версионного контроля, вносить изменения без риска.

14. Проверить состояние локального репозитория Git можно с помощью команды `git status`.

15. При добавлении/изменении файла в локальный репозиторий Git – файл появляется как `untracked` (новый) или `modified` (измененный). При добавлении нового/измененного файла под версионный контроль с помощью команды `git add` – файл становится `staged` (готов к коммиту). При фиксации изменений с помощью команды `git commit` – изменения фиксируются в локальном репозитории. При отправке изменений на сервер с помощью команды `git push` – коммиты отправляются на удаленный репозиторий, синхронизируя локальный и удаленный репозиторий.

16. Действия при работе на компьютере 1: клонирование репозитория (`git clone`), работа с файлами, добавление изменений в индекс (`git add`), создание коммита (`git commit`), отправляем изменения на GitHub (`git push`). Действия при работе на компьютере 2: клонирование репозитория (`git clone`), синхронизация перед началом работы (`git pull`), работа с файлами, добавление изменения в индекс (`git add`), создание коммита (`git commit`), синхронизация с GitHub (`git pull`), решение возможных конфликтов и отправка изменений (`git push`). В итоге, оба компьютера всегда делают `git pull` перед началом работы, чтобы получить последние изменения.

17. Популярные сервисы работающие с Git: GitLab, Bitbucket, SourceForge, AWS CodeCommit, Azure DevOps Repos.

Таблица 1. Сравнительный анализ GitHub и GitLab

Функция	GitHub	GitLab
Репозитории	Публичные и приватные	Публичные и приватные
CI/CD	GitHub Actions	Встроено
Управление проектами	Kanban-доски, Issues, Projects	Более развитые инструменты
Размещение	Только облако GitHub	Облако или собственный сервер
Сообщество	Самое крупное	Меньше, но активно используется в компаниях

18. Популярные графические клиенты для Git: GitKraken, Sourcetree, GitHub Desktop, SmartGit, TortoiseGit. В лабораторной работе были выполнены клонирование репозитория, добавление новых файлов (GitHub Desktop автоматически отображает новые или изменённые файлы в списке Changes.), создание коммита (в разделе Changes: Отмечаем файлы для коммита. Вводим сообщение коммита в поле Summary. Нажимаем Commit to main), отправка изменения на GitHub (после создания коммита кнопка Push origin становится активной, нажимаем и коммиты отправляются на GitHub).

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