Лабораториая работа №2

Отчет

Устинова Виктория Вадимовна

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Изучить идеологию и применение средств контроля версий. Освоить умения по работе с git.

2 Задание

Создать базовую конфигурацию для работы с git. Создать ключ SSH. Создать ключ PGP. Настроить подписи git. Зарегистрироваться на Github. Создать локальный каталог для выполнения заданий по предмету.

3 Выполнение лабораторной работы

Установка программного обеспечения git и gh(рис. 3.1).

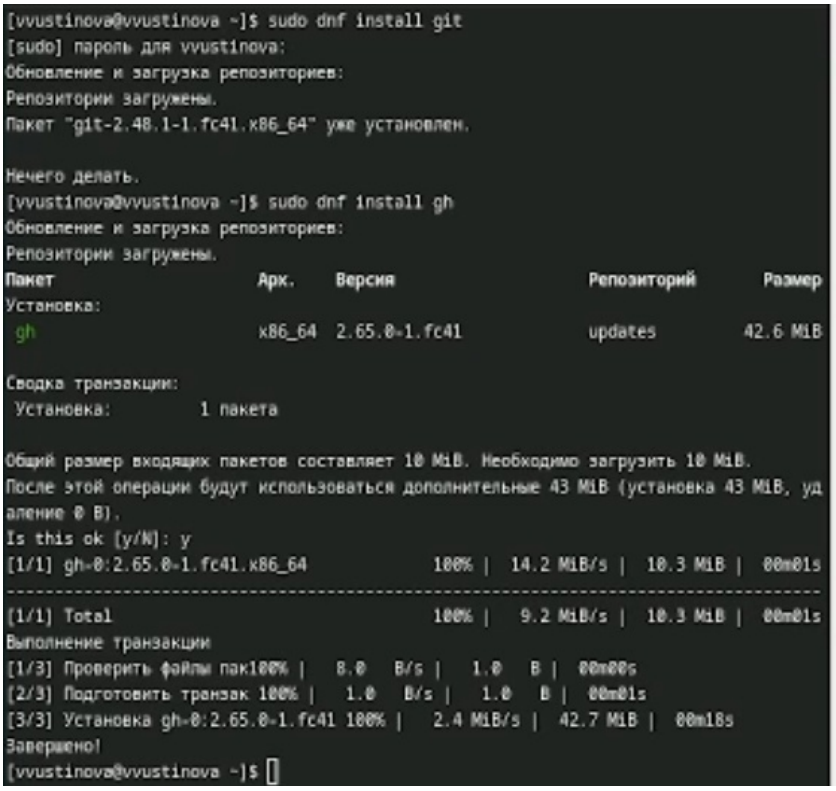

Рис. 3.1: Процесс установки

Зададим имя и email владельца репозитория и настроим utf-8 в выводе сообщений git(рис. 3.2).

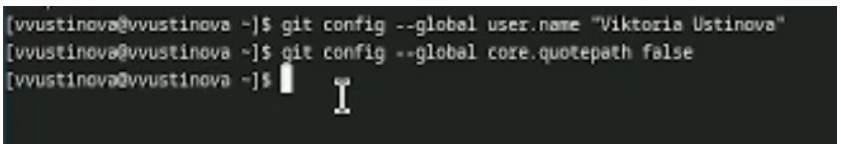

Рис. 3.2: задаем имя и настраиваем utl-8

Зададим имя начальной ветки (будем называть её master)(рис. 3.3).

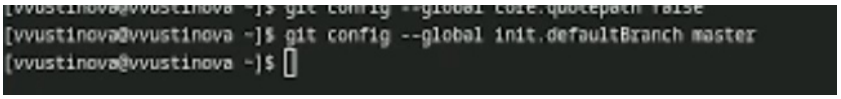

Рис. 3.3: задаем имя ветке

Параметр autocrlf и параметр safecrlf(рис. 3.4).

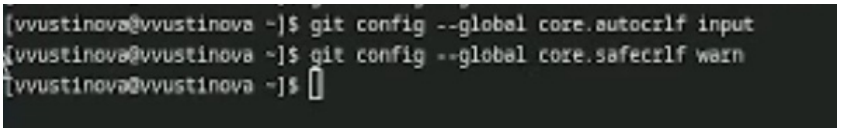

Рис. 3.4: пишем параметры

Создаем клюс ssh по алгоритму rsa с ключём размером 4096 бит(рис. 3.5).

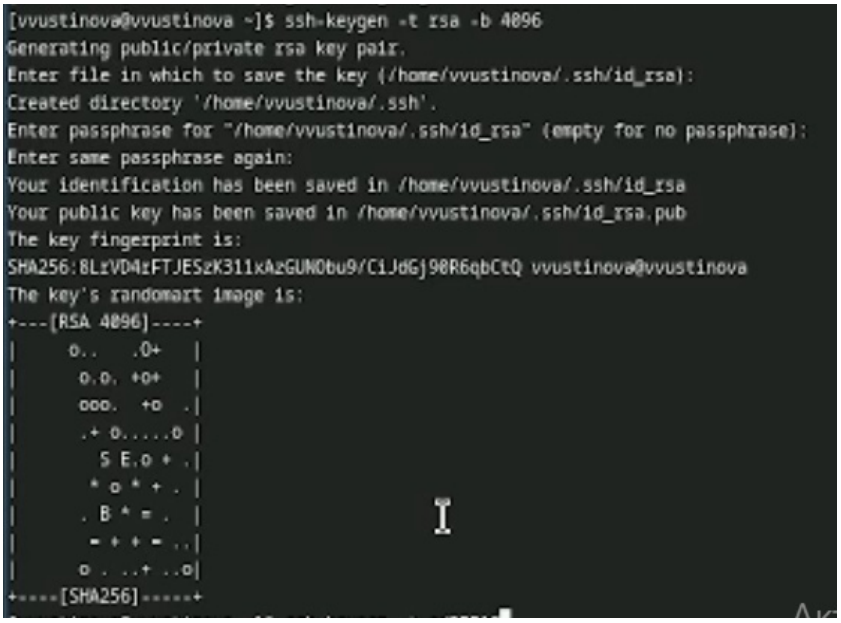

Рис. 3.5: вводим команду

Создаем клюс ssh по алгоритму ed25519(рис. 3.6).

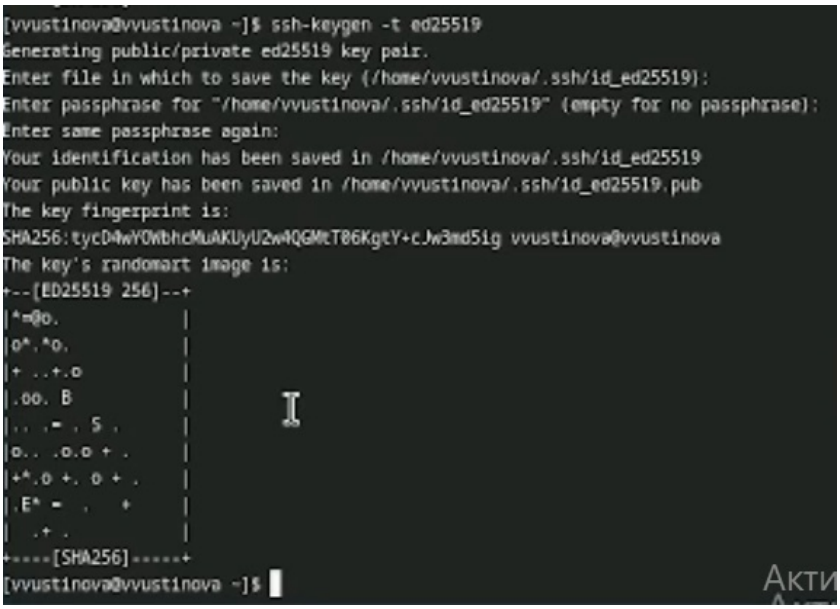

Рис. 3.6: Вводим команду

Генерируем ключ pgp(рис. 3.7).

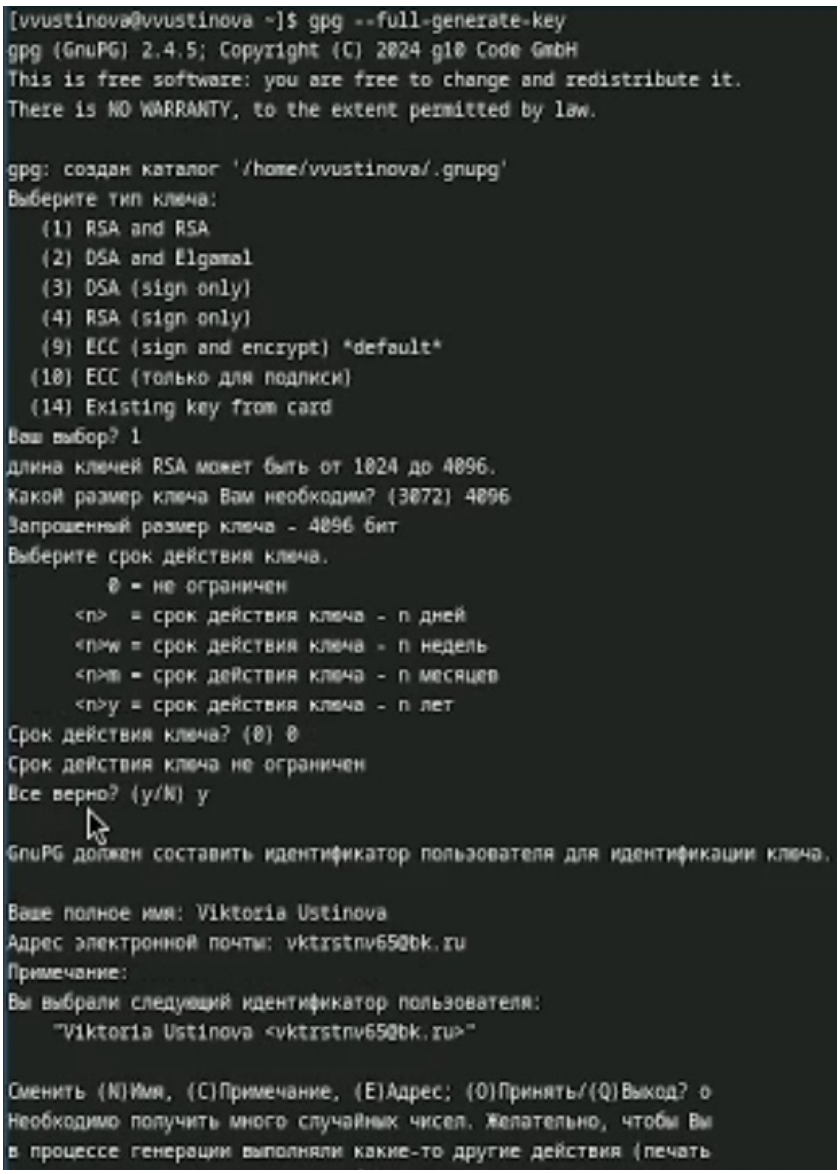

Рис. 3.7: Генерируем ключ командой и выбираем опции

Выводим список ключей и копируем отпечаток приватного ключа(рис. 3.8).

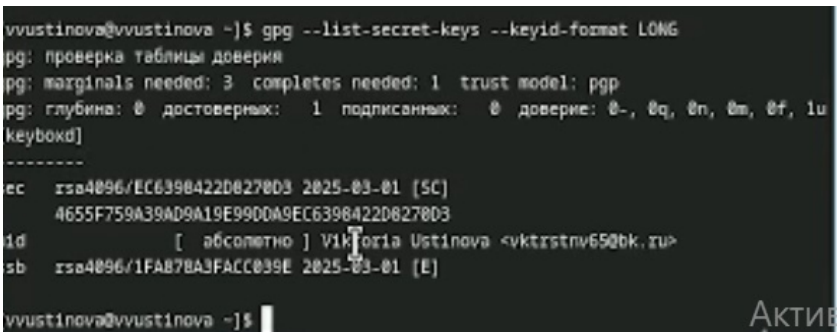

Рис. 3.8: Копируем отпечаток это почта

Скопируйте ваш сгенерированный PGP ключ в буфер обмена(рис. 3.9).

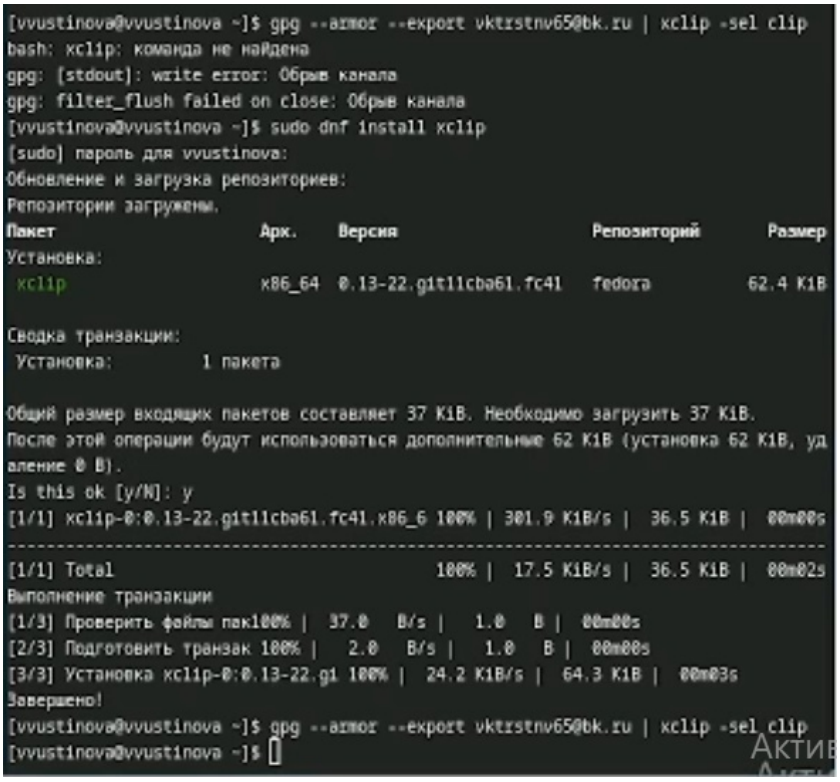

Рис. 3.9: Копируем ключ и устанавливаем команду xclip

Переходим в гитхаб и находим gpg ключ(рис. 3.10).

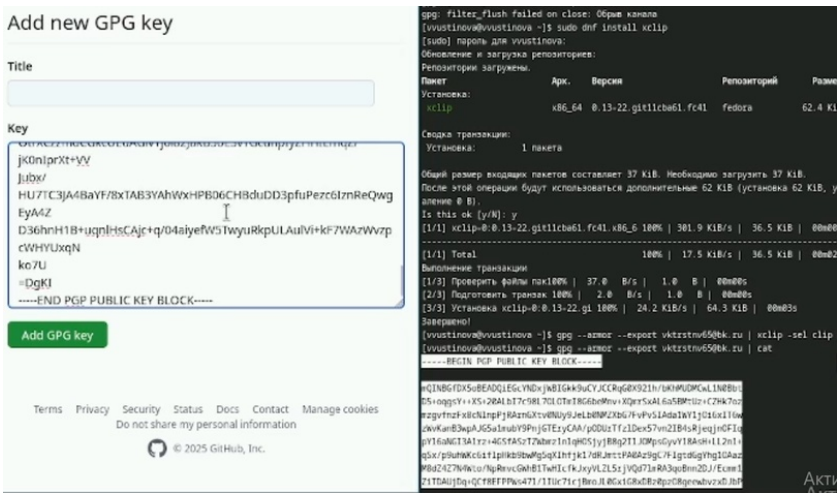

Рис. 3.10: Вставляем ключ в гитхаб

Используя введёный email, укажите Git применять его при подписи коммитов(рис. 3.11).

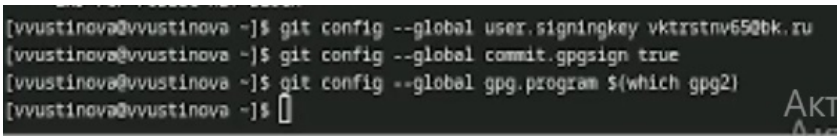

Рис. 3.11: Настраиваем подписи

Для начала необходимо авторизоваться(рис. 3.12).

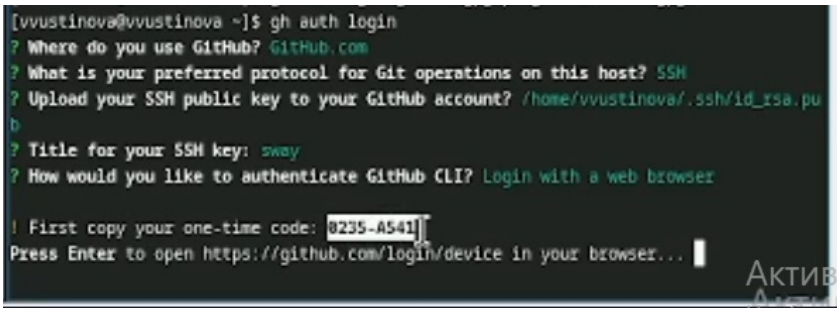

Рис. 3.12: Авторизовываемся, чтобы настроить gh

Копируем код,выделенный на предыдущем фото и вставляем в окно(рис. 3.13).

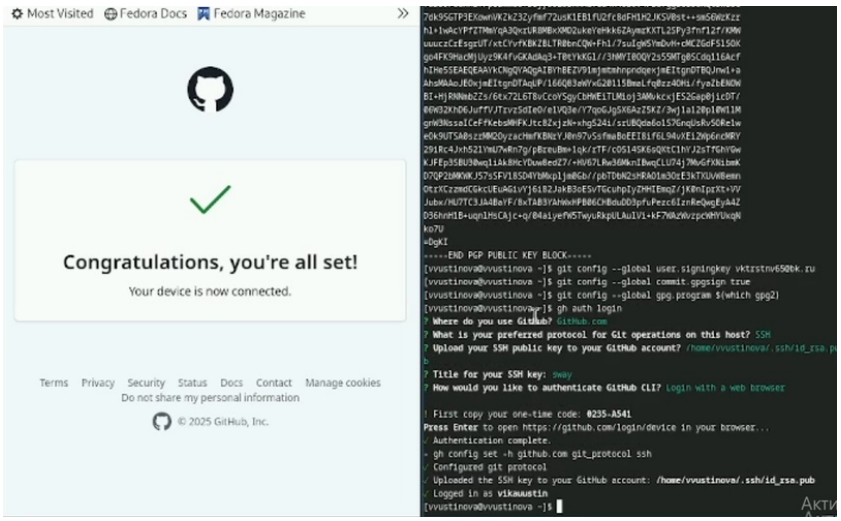

Рис. 3.13: Получилось успешно авторизоваться

Необходимо создать шаблон рабочего пространства(рис. 3.14).

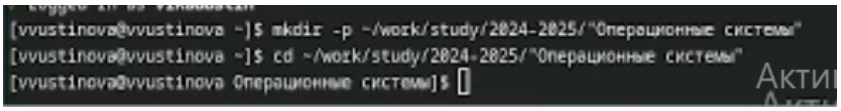

Рис. 3.14: Создаем шаблон используя mkdir и переходим в него cd

Создаем репозиторий и клонируем все на гитхаб(рис. 3.15).

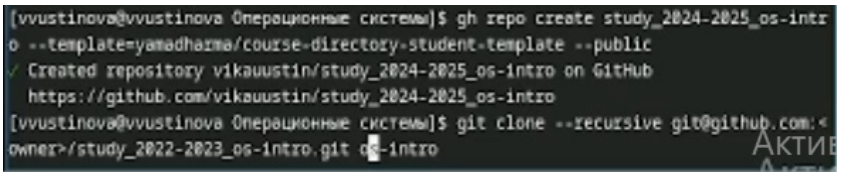

Рис. 3.15: Вводим команды из туиса

Переходим в каталог курса и создаем необходимые каталоги(рис. 3.16).

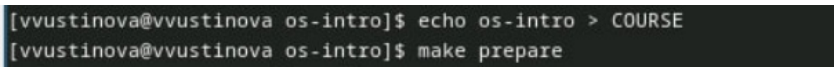

Рис. 3.16: Вводим команды

Отправляем все файлы на гитхаб(рис. 3.17).

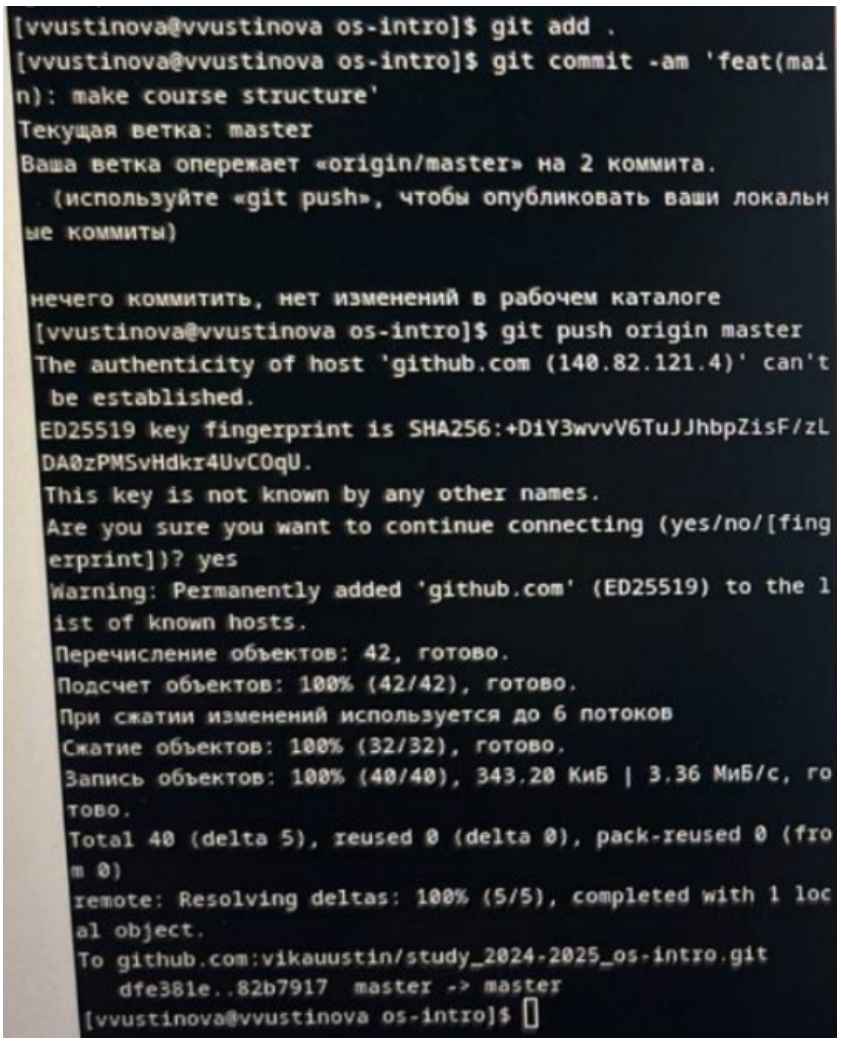

Рис. 3.17: Получилось успешно отправить

4 Выводы

У нас получилось изучить идеологию и применение средств контроля версий и освоить умения по работе с git.

5 Ответы на контольные вопросы

1. Система контроля версий (Version Control System, VCS) – это инструмент, который отслеживает изменения в наборе файлов с течением времени. Представьте себе возможность вернуться к любой предыдущей версии документа, программы или веб-сайта, посмотреть, кто, когда и почему внес изменения.
2. VCS позволяет нескольким разработчикам работать над одним проектом одновременно.Если что-то пошло не так, VCS позволяет быстро вернуться к более ранней версии файла или всего проекта.VCS позволяет создавать отдельные ветки разработки, чтобы экспериментировать с новыми функциями или исправлять ошибки, не затрагивая основную линию разработки.Хранилище (Repository): Это центральное место, где хранится вся история проекта.Commit: Это фиксация набора изменений в хранилище. История (History): Это последовательность всех commit-ов, выполненных в проекте, упорядоченных во времени. История позволяет видеть эволюцию проекта и отслеживать все изменения.Рабочая копия (Working Copy): Это локальная копия файлов проекта на компьютере разработчика.
3. Централизованные VCS: В централизованных VCS есть одно центральное хранилище, где хранится вся история проекта. Разработчики получают рабочую копию из этого хранилища, вносят изменения и отправляют их обратно в центральное хранилище. Примеры: Subversion (SVN), CVS.Децентрализованные VCS: В децентрализованных VCS каждый разработчик имеет полную копию хранилища со всей историей проекта. Разработчики могут

работать независимо друг от друга и обмениваться изменениями напрямую, минуя центральный сервер.Примеры: Git, Mercurial.

4. 1. Инициализация хранилища: Создается новое хранилище для проекта (например, с помощью git init).2. Добавление файлов: Файлы проекта добавляются в область отслеживания VCS.3. Фиксация изменений (commit): Внесенные изменения фиксируются в хранилище с комментарием, описы я, удаляются. 4. Повторение шагов 3 и 4: Процесс внесения изменений и их фиксации повторяется по мере необходимости. 5. Просмотр истории: Просматривается история изменений, чтобы понять, что и когда было изменено. 6. Возврат к предыдущим версиям: При необходимости выполняется возврат к одной из предыдущих версий файла или проекта.

5. 1. Клонирование хранилища: Разработчик клонирует удаленное хранилище на свой компьютер, создавая локальную рабочую копию. 2. Создание ветки (branch): (Опционально) Создается отдельная ветка для внесения изменений, чтобы не затрагивать основную линию разработки. 3. Внесение изменений: Разработчик вносит изменения в локальную рабочую копию. 4. Фиксация изменений (commit): Изменения фиксируются в локальном хранилище. 5. Отправка изменений (push): Локальные изменения отправляются в удаленное хранилище (обычно в свою ветку). 6. Создание запроса на слияние (pull request): Разработчик создает запрос на слияние своих изменений из ветки в основную ветку. 7. Обзор изменений (code review): Другие разработчики просматривают изменения и оставляют комментарии. 8. Слияние изменений (merge): После одобрения изменений они сливаются в основную ветку. 9. Разрешение конфликтов: Если при слиянии возникают конфликты (например, два разработчика изменили один и тот же участок кода), они разрешаются вручную. 10. Синхронизация (pull): Разработчик получает последние изменения из удаленного хранилища в свою локаль-

ную рабочую копию.

6. Основные задачи, решаемые Git: • Контроль версий: Git отслеживает изменения в файлах и позволяет возвращаться к предыдущим версиям. • Совместная работа. • Ветвление и слияние: Git позволяет создавать отдельные ветки разработки для экспериментов и исправления ошибок, а затем сливать эти ветки обратно в основную линию разработки. • Распределенная разработка: Git - это децентрализованная VCS, поэтому каждый разработчик имеет полную копию хранилища.
7. Основные команды Git:git init: Инициализирует новый Git-репозиторий.git clone : Клонирует существующий Git-репозиторий с удаленного сервера.git add : Добавляет файлы в область подготовки (staging area).git commit -m “Сообщение”: Фиксирует изменения из области подготовки в локальном репозитории.git status: Показывает текущий статус файлов в репозитории.git branch: Управляет ветками (создание, удаление, переключение).git push : Отправляет локальные коммиты в удаленный репозиторий.git pull : Получает изменения из удаленного репозитория и сливает их в текущую ветку. git revert: Создает новый коммит, отменяющий изменения из предыдущего коммита. git stash: Временно сохраняет изменения в рабочей директории, чтобы переключиться на другую ветку.
8. Локальный: mkdir my_project cd my_project git init Создаем файл hello.txt echo “Hello, world!” > hello.txt Добавляем файл в область подготовки git add hello.txt Фиксируем изменения git commit -m “Initial commit: Added hello.txt” Удаленный:git remote add origin Отправляем локальную ветку master в удаленную ветку master git push origin master Клонируем репозиторий с GitHub git clone
9. Ветвь – это указатель на определенный commit в истории проекта. Ветви позволяют работать над разными функциями или исправлениями ошибок параллельно, не затрагивая основную линию разработки. Зачем нужны ветви: 1.Изоляция изменений. 2.Параллельная разработка. каждая в своей

ветке. 3.Эксперименты. 4.Подготовка к релизу.

10. Зачем это нужно 1.Очистка репозитория: Предотвращает засорение репозитория ненужными файлами. 2.Безопасность: Предотвращает попадание конфиденциальных данных (например, паролей) в репозиторий. 3.Производительность: Уменьшает размер репозитория и ускоряет операции Git.