Лабораторная работа №2

НПИбд-03-23

Дмитриев Владимир Константинович

Содержание

# 1 Цель работы

Изучить идеологию и применение средств контроля версий. Приобрести практические навыки по работе с системой 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

На первом этапе выполнения лабораторной работы необходимо произвести установку и базовую настройку git. Это было сделано ранее, при выполнение лабораторной работы №3 на курсе “Архитектура компьютера” (рис. [[1](#fig:001)]).

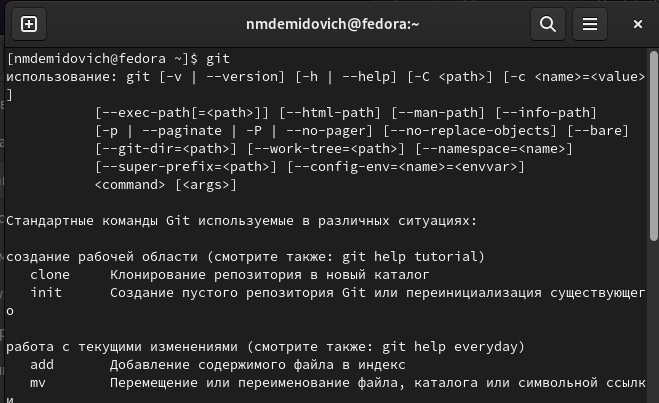


Figure 1: Команда git в терминале

Далее я приступил к процессу создания нового репозитория для работ по курсу “Операционные системы”. И для начала мною был сгенерирован новый ssh-ключ (рис. [[2](#fig:002)]) и добавлен на github (рис. [[4](#fig:004)]).

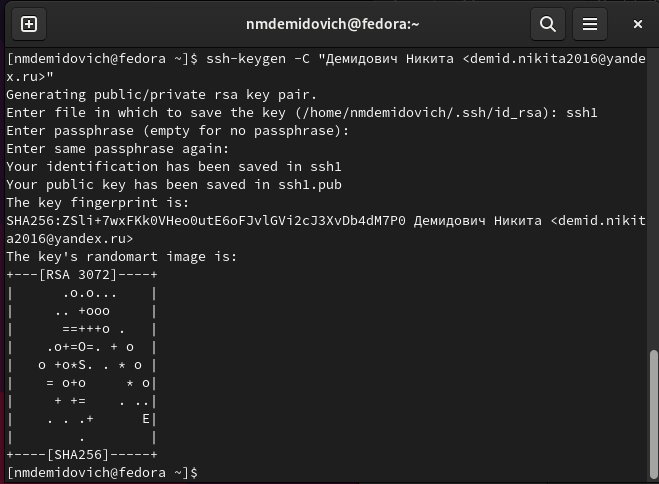


Figure 2: Начало процесса генерации нового ssh-ключа

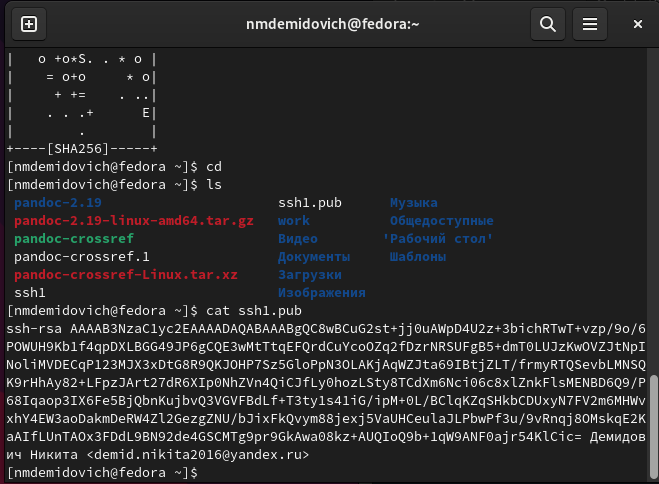


Figure 3: Содержимое нового файла ssh1.pub

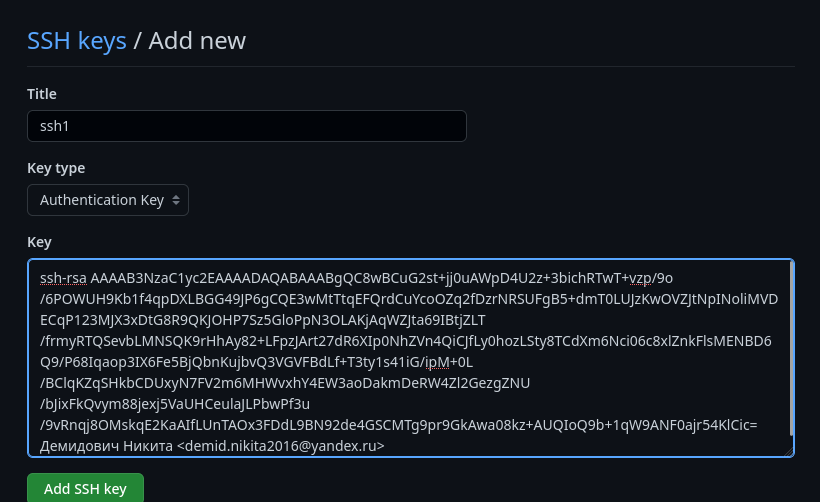


Figure 4: Добавление нового ssh-ключа на github

Далее я создал gpg-ключ (рис. [[5](#fig:005)]).

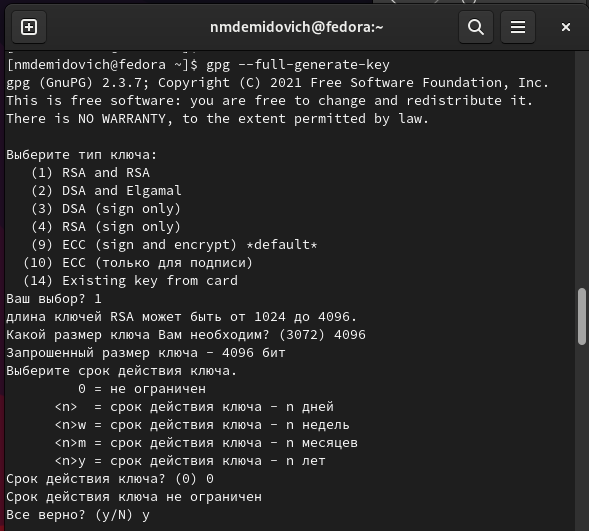


Figure 5: Начало процесса генерации нового gpg-ключа

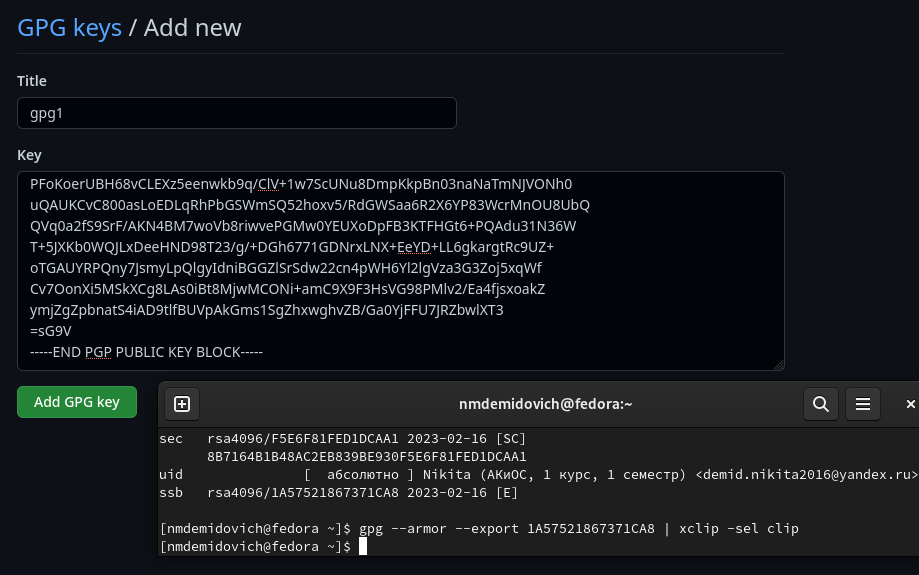


Figure 6: Добавление нового gpg-ключа на github

*P.S: В описании была допущена ошибка: семестр не первый, а второй (уже исправлено).*

Затем мною была произведена настройка автоматических подписей коммитов git (рис. [[7](#fig:007)]).

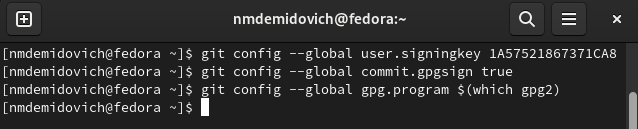


Figure 7: Добавление нового gpg-ключа на github

И создан репозиторий курса на основе шаблона (рис. [[9](#fig:009)]).

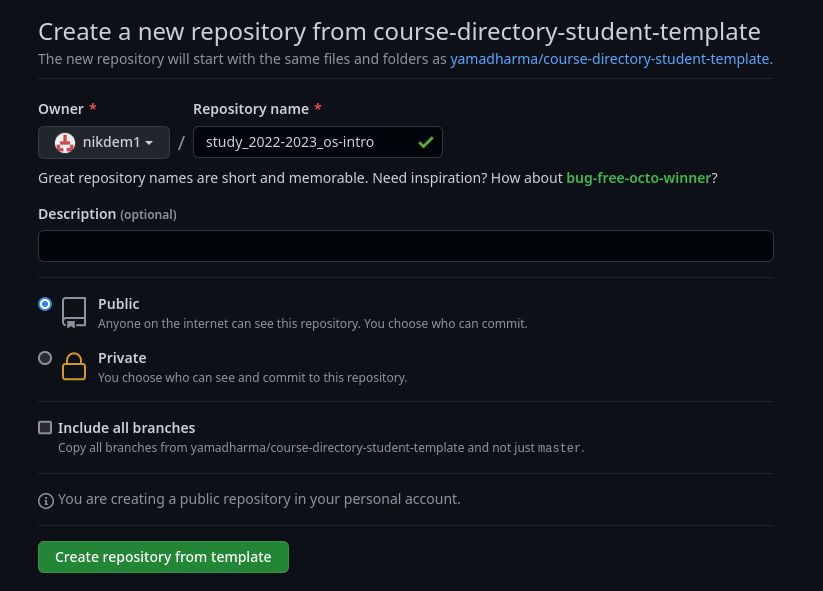


Figure 8: Копирование репозитория на сайте github

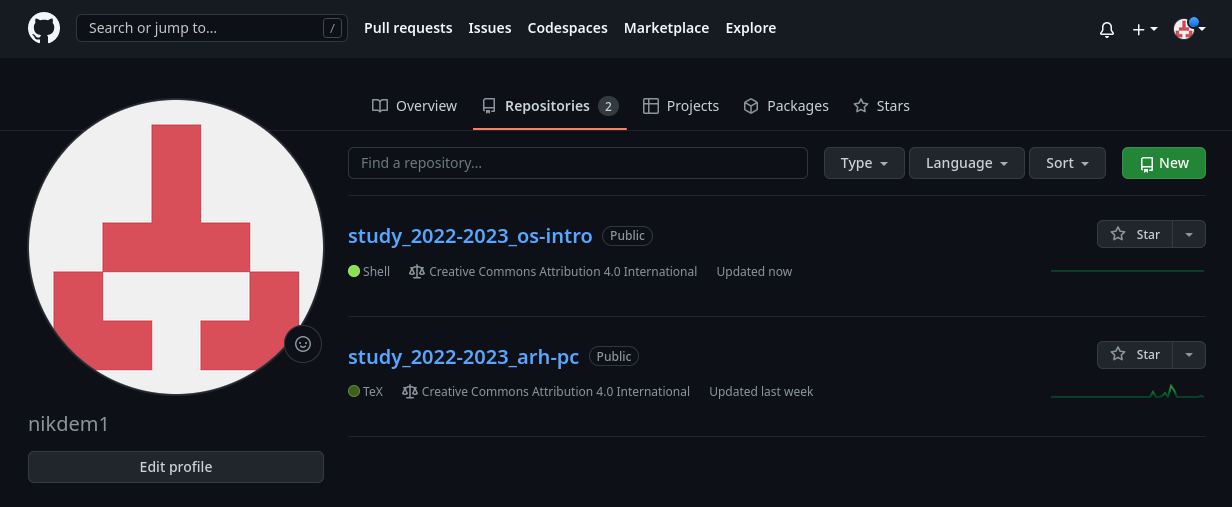


Figure 9: Репозитории на github

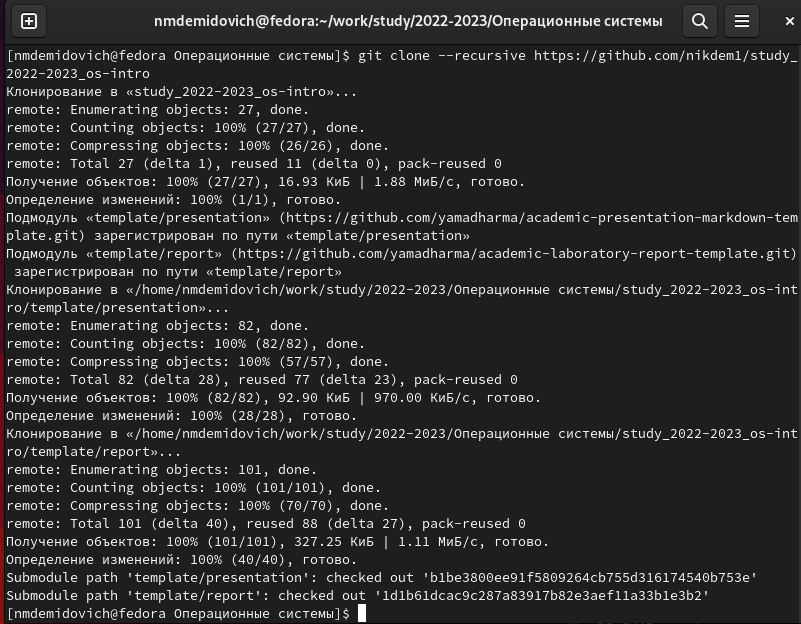


Figure 10: Создание каталога курса и его настройка

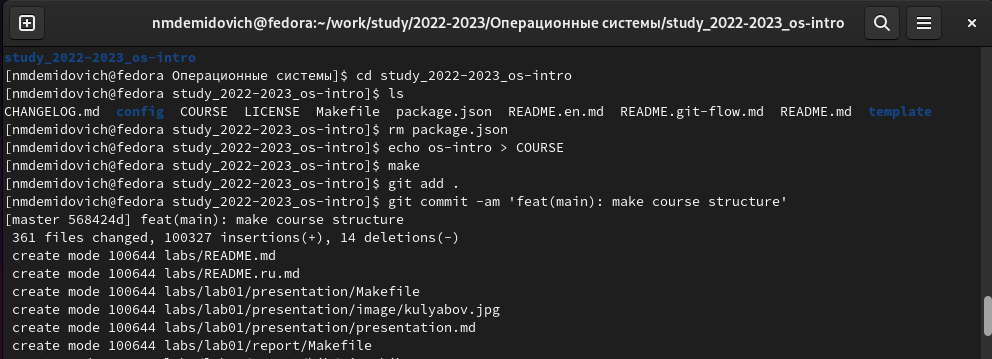


Figure 11: Настройка каталога курса

После этого я создал персональный токен для авторизации в терминале и отправил файлы на сервер (рис. [[13](#fig:013)])

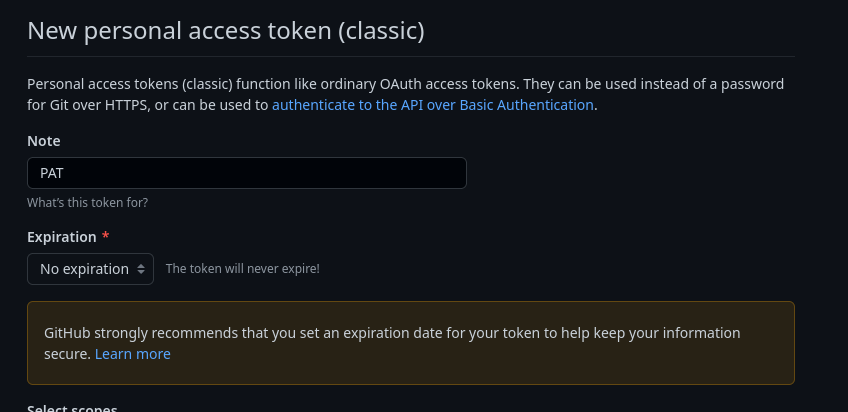


Figure 12: Создание персонального токена для авторизации

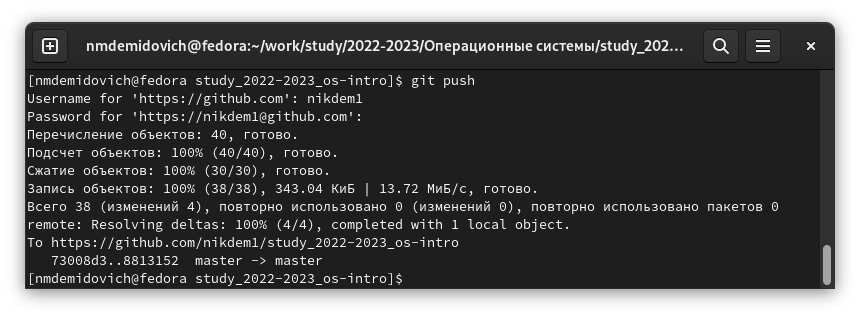


Figure 13: Авторизация с помощью токена и отправка файлов на сервер

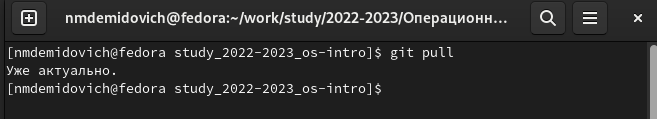


Figure 14: Проверка выгрузки содержимого локального репозитория

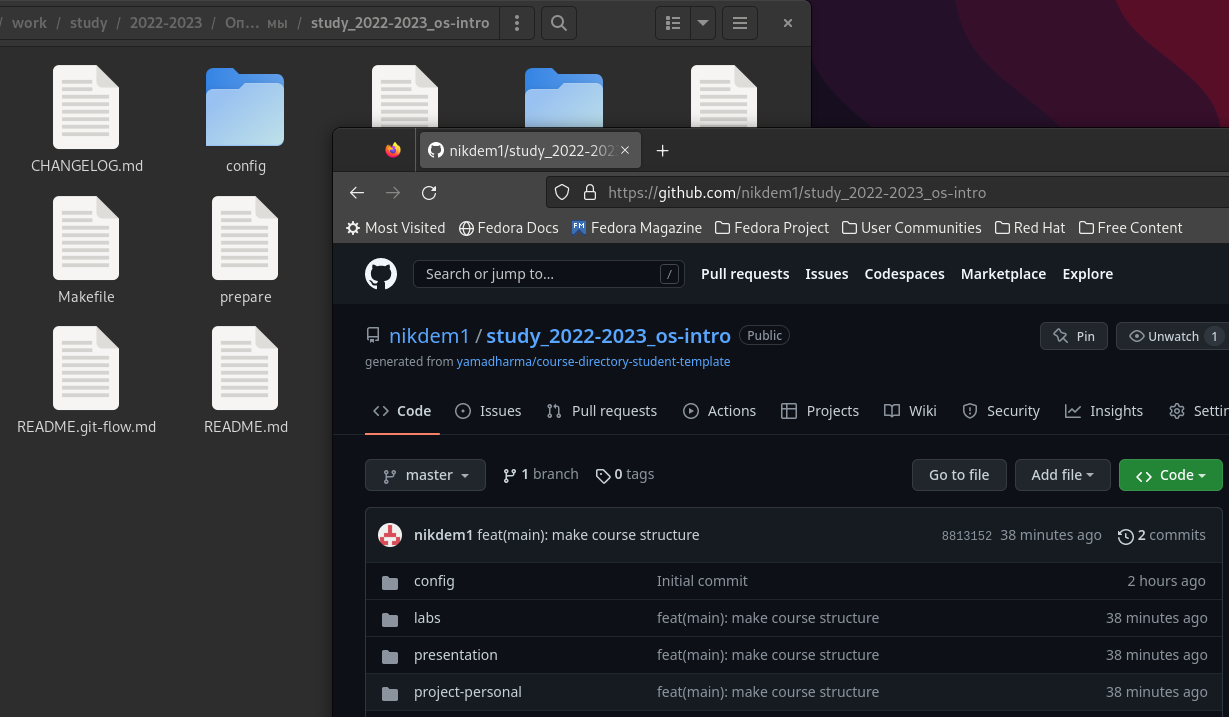


Figure 15: Локальный репозиторий и репозиторий на github

# 4 Контрольные вопросы

**1. Что такое системы контроля версий (VCS) и для решения каких задач они предназначаются?**

Система контроля версий (VCS) - программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое. Такие системы наиболее широко используются при разработке программного обеспечения для хранения исходных кодов разрабатываемой программы. Однако они могут с успехом применяться и в других областях, в которых ведётся работа с большим количеством непрерывно изменяющихся электронных документов. 2. Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия. Хранилище – репозиторий - место хранения всех версий и служебной информации. Commit - это команда для записи индексированных изменений в репозиторий. История – место, где сохраняются все коммиты, по которым можно посмотреть данные о коммитах. Рабочая копия – текущее состояние файлов проекта, основанное на версии, загруженной из хранилища. 3. Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида.

Централизованные системы – это системы, в которых одно основное хранилище всего проекта, и каждый пользователь копирует необходимые ему файлы, изменяет и вставляет обратно. Пример – Subversion. Децентрализованные системы – система, в которой каждый пользователь имеет свой вариант репозитория и есть возможность добавлять и забирать изменения из репозиториев. Пример – Git.

**2. Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия.**

Система контроля версий (Version Control System, VCS) — это программное обеспечение для облегчения работы с изменяющейся информацией. Репозиторий (или хранилище) - в нем хранятся все документы вместе с историей их изменения и другой служебной информацией. Тэг commint позволяет сохранять и измененять изменения в репозитории. Рабочая копия - копия проекта, связанная с репозиторием.

**3. Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида.**

**4. Опишите действия с VCS при единоличной работе с хранилищем.**

В рабочей копии, которую исправляет человек, появляются правки, которые отправляются в хранилище на каждом из этапов. То есть в правки в рабочей копии появляются, только если человек делает их (отправляет их на сервер) и никак по-другому.

**5. Опишите порядок работы с общим хранилищем VCS.**

Если хранилище общее, то в рабочую копию каждого, кто работает над проектом, приходят изменения, отправленные на сервер одним из команды. Рабочая правка каждого может изменяться вне зависимости от того, делает ли конкретный человек правки или нет.

**6. Каковы основные задачи, решаемые инструментальным средством git?**

У Git две основных задачи: первая — хранить информацию обо всех изменениях в вашем коде, начиная с самой первой строчки, а вторая — обеспечение удобства командной работы над кодом.

**7. Назовите и дайте краткую характеристику командам git.**

– создание основного дерева репозитория: git init – получение обновлений (изменений) текущего дерева из центрального репозитория: git pull – отправка всех произведённых изменений локального дерева в центральный репозиторий: git push – просмотр списка изменённых файлов в текущей директории: git status – просмотр текущих изменения: git diff – сохранение текущих изменений: – добавить все изменённые и/или созданные файлы и/или каталоги: git add . – добавить конкретные изменённые и/или созданные файлы и/или каталоги: git add – удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории): git rm имена\_файлов – сохранить все добавленные изменения и все изменённые файлы: git commit -am ‘Описание коммита’ – сохранить добавленные изменения с внесением комментария через встроенный редактор: git commit – создание новой ветки, базирующейся на текущей: git checkout -b имя\_ветки – переключение на некоторую ветку: git checkout имя\_ветки (при переключении на ветку, которой ещё нет в локальном репозитории, она будет создана и связана с удалённой) – отправка изменений конкретной ветки в центральный репозиторий: 1 git push origin имя\_ветки – слияние ветки с текущим деревом: 1 git merge –no-ff имя\_ветки – удаление локальной уже слитой с основным деревом ветки: git branch -d имя\_ветки – принудительное удаление локальной ветки: git branch -D имя\_ветки – удаление ветки с центрального репозитория: git push origin :имя\_ветки

**8. Приведите примеры использования при работе с локальным и удалённым репозиториями.**

Работа с удаленным репозиторием: git remote – просмотр списка настроенных удаленных репозиториев.

Работа с локальным репозиторием: git status - выводит информацию обо всех изменениях, внесенных в дерево директорий проекта по сравнению с последним коммитом рабочей ветки.

**9. Что такое и зачем могут быть нужны ветви (branches)?**

Ветка (англ. branch) — это последовательность коммитов, в которой ведётся параллельная разработка какого-либо функционала. Ветки нужны, чтобы несколько программистов могли вести работу над одним и тем же проектом или даже файлом одновременно, при этом не мешая друг другу. Кроме того, ветки используются для тестирования экспериментальных функций: чтобы не повредить основному проекту, создается новая ветка специально для экспериментов.

**10. Как и зачем можно игнорировать некоторые файлы при commit?**

Игнорируемые файлы — это, как правило, артефакты сборки и файлы, генерируемые машиной из исходных файлов в вашем репозитории, либо файлы, которые по какой-либо иной причине не должны попадать в коммиты. В Git нет специальной команды для указания игнорируемых файлов: вместо этого необходимо вручную отредактировать файл . Временно игнорировать изменения в файле можно командой git update-index-assumeunchanged.

# 5 Выводы

В результате выполнения данной лабораторной работы я изучил идеологию и применение средств контроля версий, а также приобрел практические навыки по работе с системой git.

# Список литературы

Лабораторная работа №2 (Архитектура ОС).