Лабораторная работа №2

Первоначальная настройка git

Чемоданова Ангелина Александровна

Содержание

# 1 Цель работы

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

# 2 Задание

1. Создать базовую конфигурацию для работы с git.
2. Создать ключ SSH.
3. Создать ключ PGP.
4. Настроить подписи git.
5. Зарегистрироваться на Github.
6. Создать локальный каталог для выполнения заданий по предмету.

# 3 Теоретическое введение

Системы контроля версий. Общие понятия

Системы контроля версий (Version Control System, VCS) применяются при работе нескольких человек над одним проектом. Обычно основное дерево проекта хранится в локальном или удалённом репозитории, к которому настроен доступ для участников проекта. При внесении изменений в содержание проекта система контроля версий позволяет их фиксировать, совмещать изменения, произведённые разными участниками проекта, производить откат к любой более ранней версии проекта, если это требуется.

В классических системах контроля версий используется централизованная модель, предполагающая наличие единого репозитория для хранения файлов. Выполнение большинства функций по управлению версиями осуществляется специальным сервером. Участник проекта (пользователь) перед началом работы посредством определённых команд получает нужную ему версию файлов. После внесения изменений, пользователь размещает новую версию в хранилище. При этом предыдущие версии не удаляются из центрального хранилища и к ним можно вернуться в любой момент. Сервер может сохранять не полную версию изменённых файлов, а производить так называемую дельта-компрессию — сохранять только изменения между последовательными версиями, что позволяет уменьшить объём хранимых данных.

Системы контроля версий поддерживают возможность отслеживания и разрешения конфликтов, которые могут возникнуть при работе нескольких человек над одним файлом. Можно объединить (слить) изменения, сделанные разными участниками (автоматически или вручную), вручную выбрать нужную версию, отменить изменения вовсе или заблокировать файлы для изменения. В зависимости от настроек блокировка не позволяет другим пользователям получить рабочую копию или препятствует изменению рабочей копии файла средствами файловой системы ОС, обеспечивая таким образом, привилегированный доступ только одному пользователю, работающему с файлом.

Системы контроля версий также могут обеспечивать дополнительные, более гибкие функциональные возможности. Например, они могут поддерживать работу с несколькими версиями одного файла, сохраняя общую историю изменений до точки ветвления версий и собственные истории изменений каждой ветви. Кроме того, обычно доступна информация о том, кто из участников, когда и какие изменения вносил. Обычно такого рода информация хранится в журнале изменений, доступ к которому можно ограничить.

В отличие от классических, в распределённых системах контроля версий центральный репозиторий не является обязательным.

Среди классических VCS наиболее известны CVS, Subversion, а среди распределённых — Git, Bazaar, Mercurial. Принципы их работы схожи, отличаются они в основном синтаксисом используемых в работе команд.

Примеры использования git

Система контроля версий Git представляет собой набор программ командной строки. Доступ к ним можно получить из терминала посредством ввода команды git с различными опциями.  
Благодаря тому, что Git является распределённой системой контроля версий, резервную копию локального хранилища можно сделать простым копированием или архивацией.

Основные команды git

Перечислим наиболее часто используемые команды 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 изменений:  
  
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 имена\_файлов  
  
 удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории):  
  
 git rm имена\_файлов  
  
Сохранение добавленных изменений:  
  
 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:  
  
 git push origin имя\_ветки  
  
 слияние ветки с текущим деревом:  
  
 git merge --no-ff имя\_ветки  
  
Удаление 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

# 4 Выполнение лабораторной работы

Для начала регистрируемся на гитхабе. (рис. [[1](#fig:001)]).

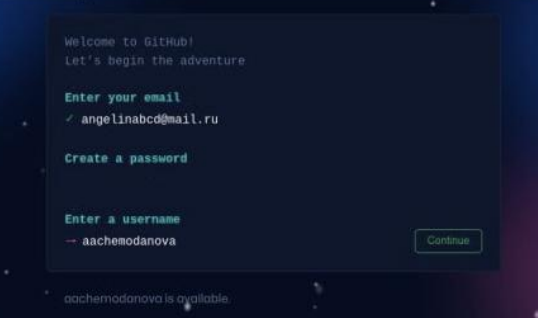


Figure 1: Регистрация на GitHub

Затем необходимо установить git-flow. (рис. [[2](#fig:002)]).

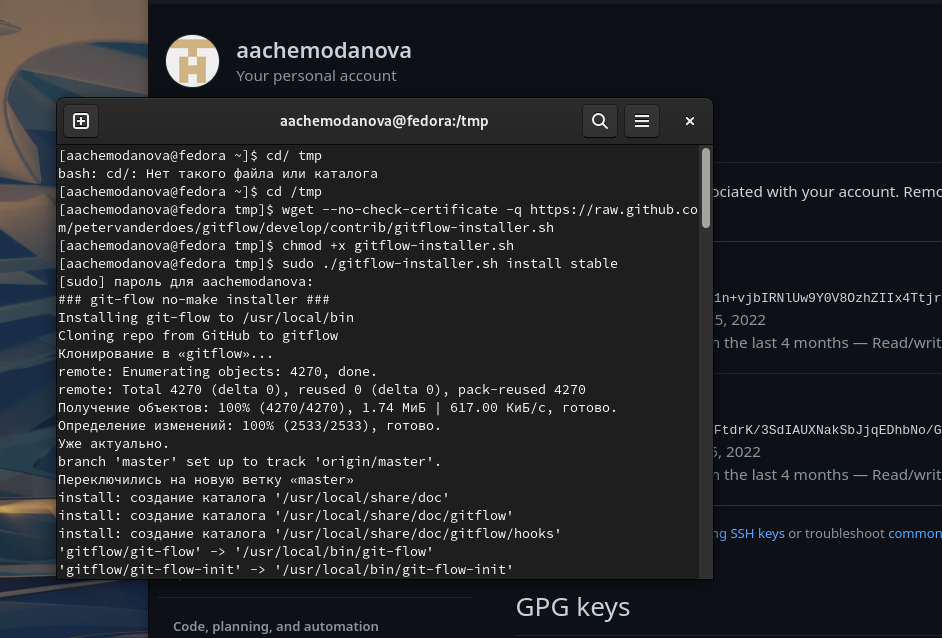


Figure 2: Установка git-flow

Устанавливаем gh. (рис. [[3](#fig:003)]).

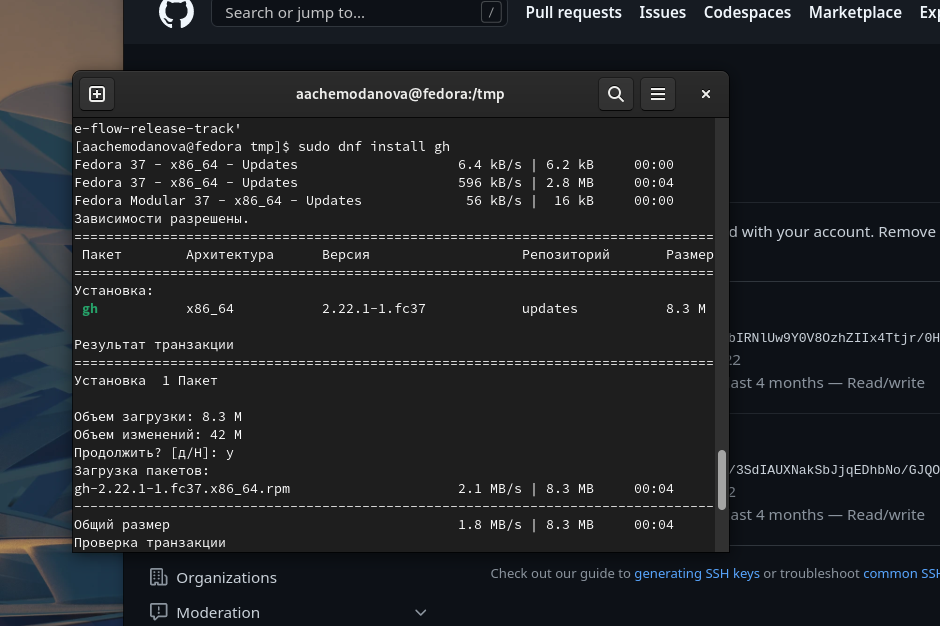


Figure 3: Установка gh

Перейдем к первичной настройке параметров git. (рис. [[4](#fig:004)]).

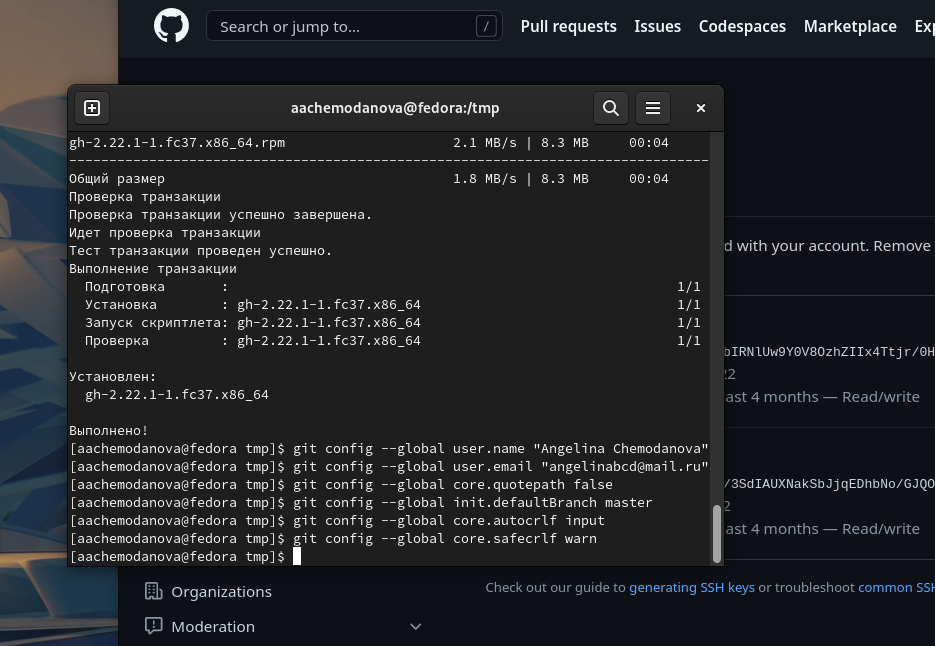


Figure 4: Первичная настройка параметров git

Создаем ключ ssh по алгориттму rsa с ключем размером 4096 бит. (рис. [[5](#fig:005)]).

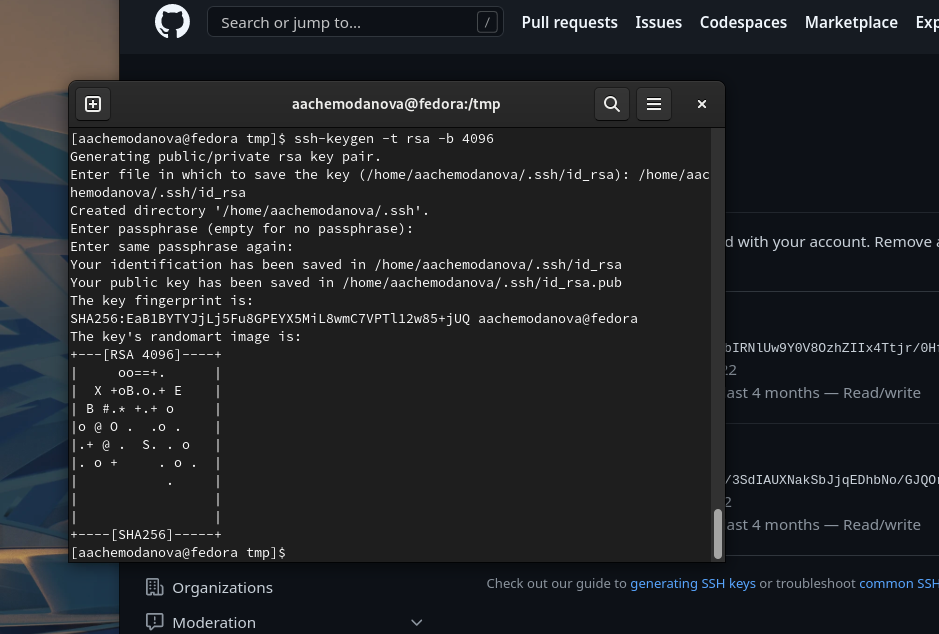


Figure 5: Создание ключа ssh по алгоритму rsa с ключем размером 4096 бит

Создаем ключ ssh по алгоритму ed25519. (рис. [[6](#fig:006)]).

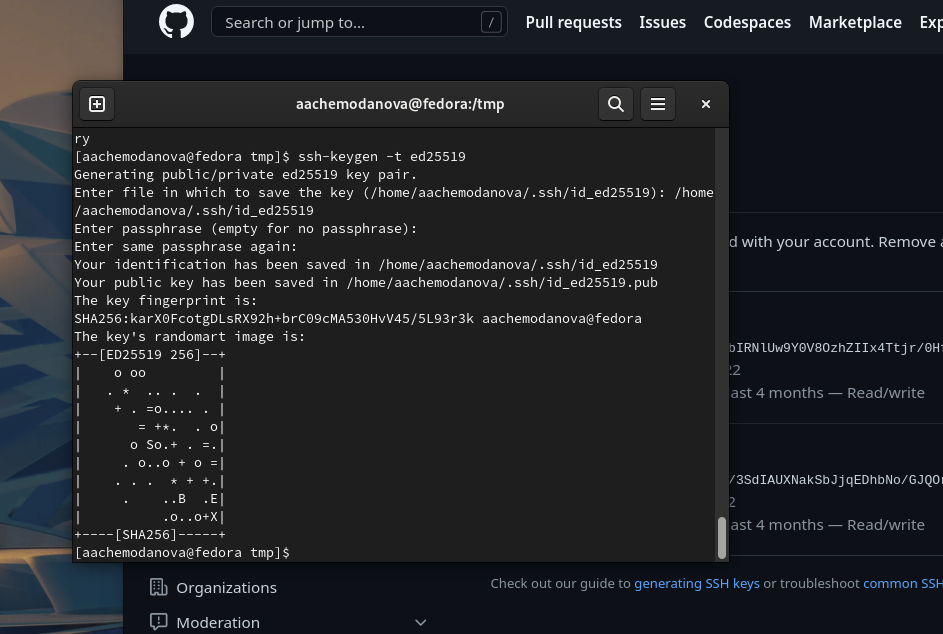


Figure 6: Создание ключа ssh по алгоритму ed25519

Создаем ключ gpg. Отвечаем на вопросы, которые заданы после введения команды. (рис. [[7](#fig:007)]).

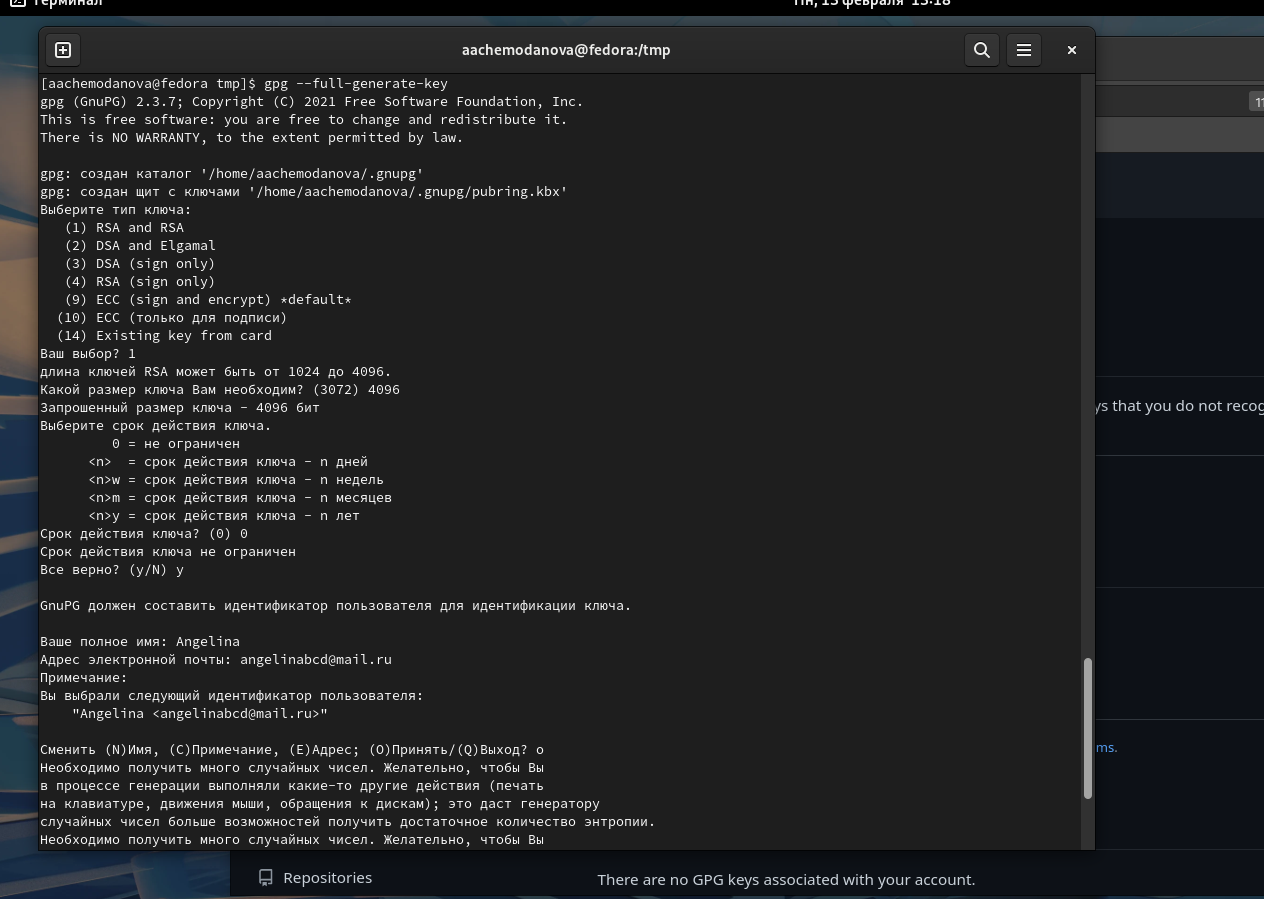


Figure 7: Создание ключа gpg

Выведем список ключей и добавим gpg-ключ на GitHub. (рис. [[8](#fig:008)]).

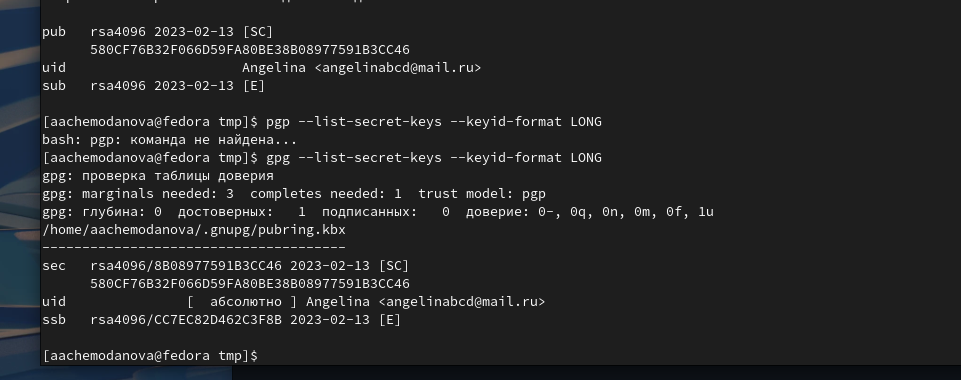


Figure 8: Отпечаток приватного ключа

Добавляем pgp-ключ на гитхаб. (рис. [[9](#fig:009)]).

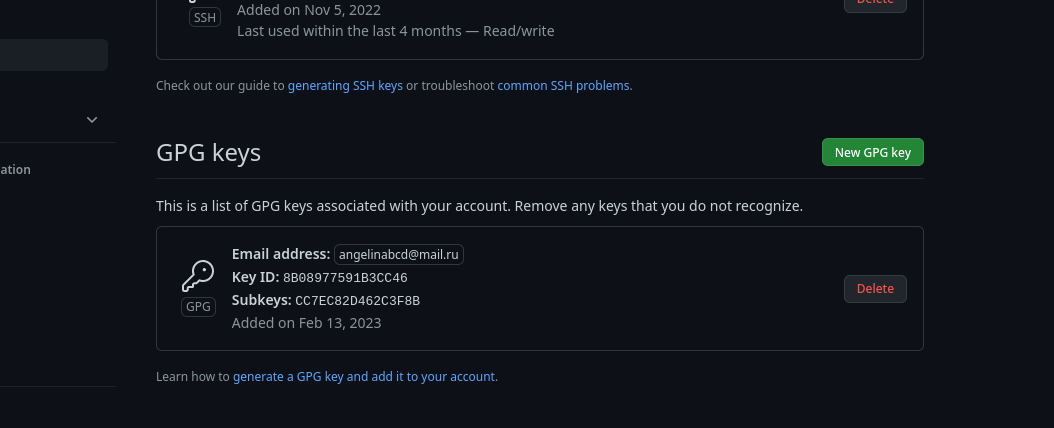


Figure 9: Добавление pgp-ключ на гитхаб

Настраиваем автоматическую подпись коммитов git. (рис. [[10](#fig:010)]).

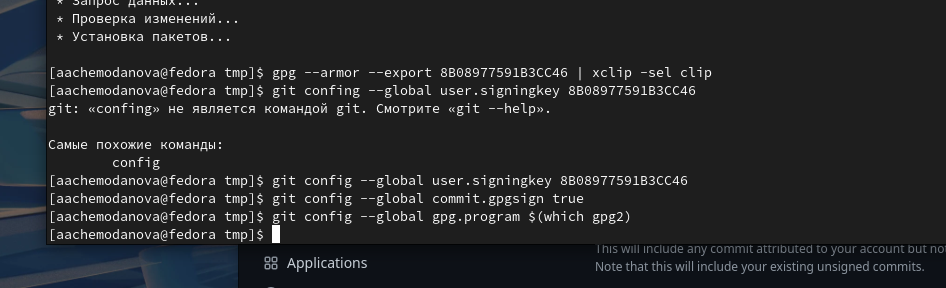


Figure 10: Настройка автоматических подписей коммитов git

Настройка gh и подтверждение авторизации. (рис. [[11](#fig:011)]).

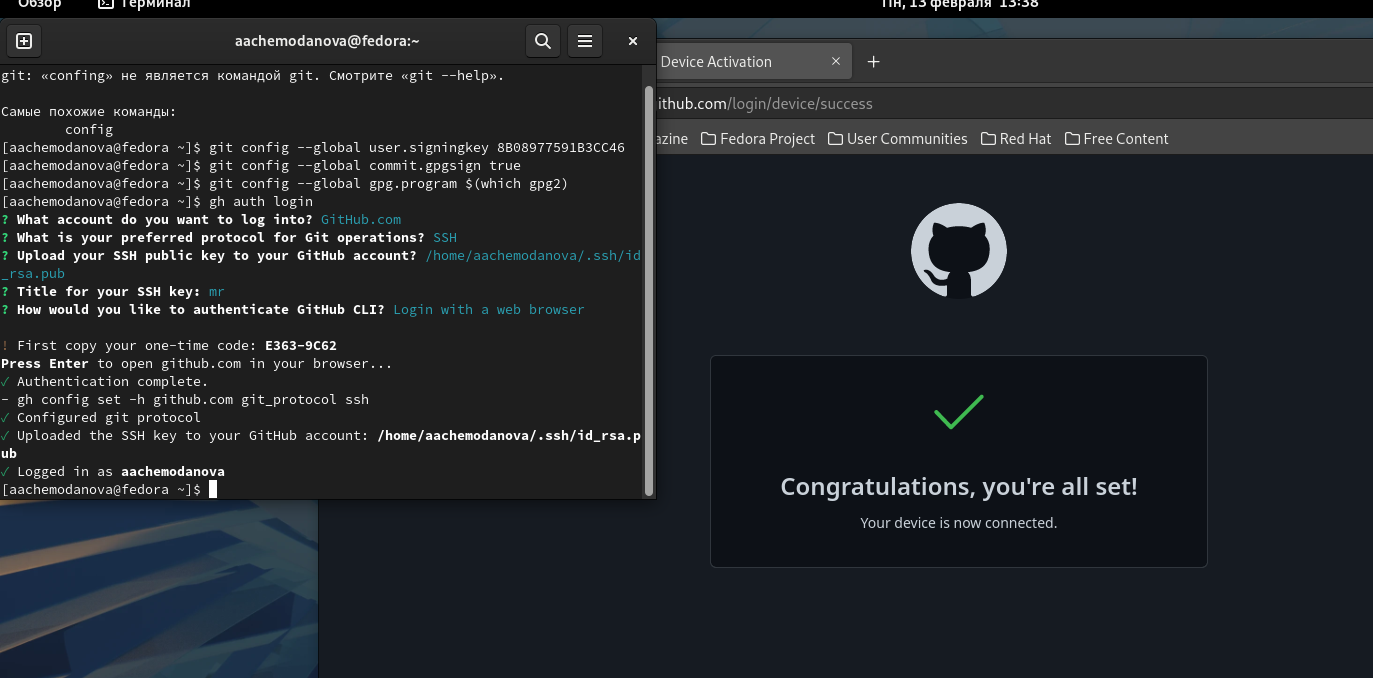


Figure 11: Настройка gh и подтверждение авторизации

Добавление ssh-ключа на гитхаб. (рис. [[12](#fig:012)]).

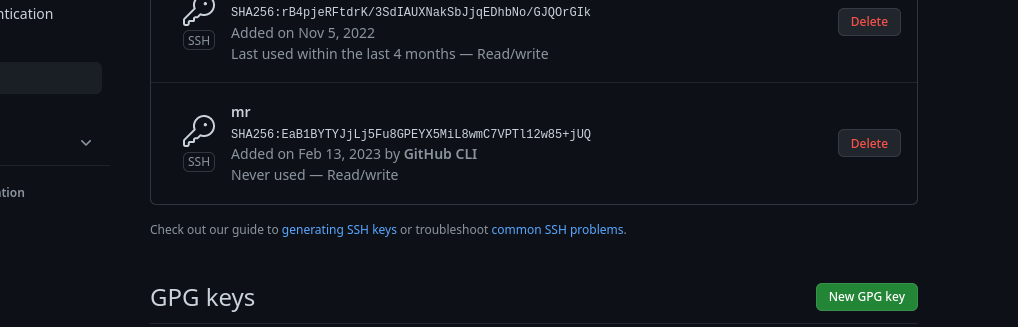


Figure 12: Добавление ssh-ключа на гитхаб

Создание репозитория курса на основе шаблона. (рис. [[13](#fig:013)]).

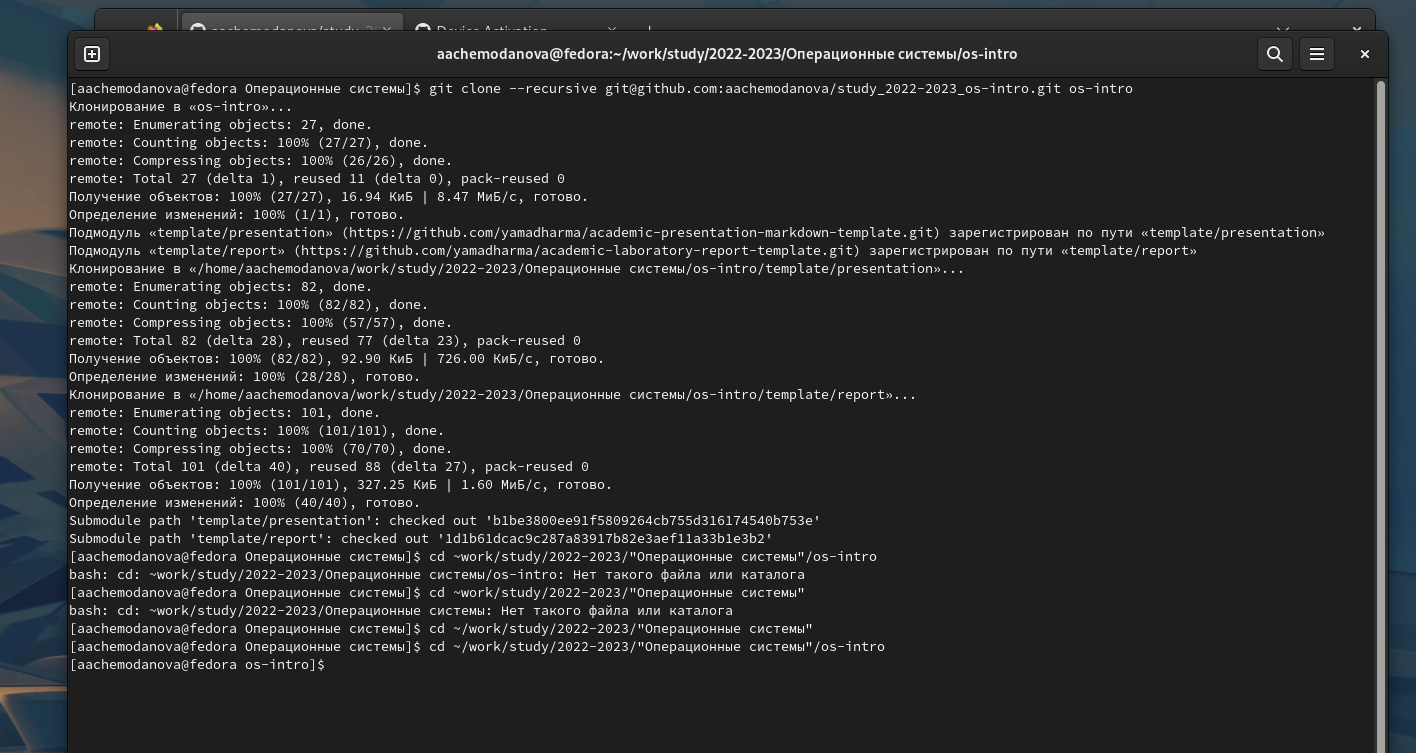


Figure 13: Создание репозитория курса

Настройка каталога курса. (рис. [[14](#fig:014)]).

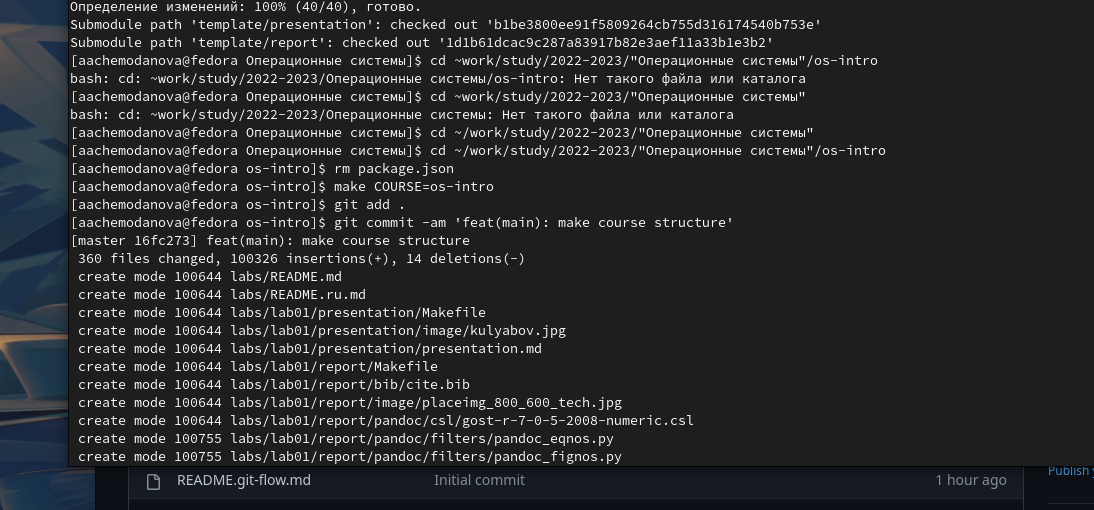


Figure 14: Настройка каталога курса

(рис. [[15](#fig:015)]).

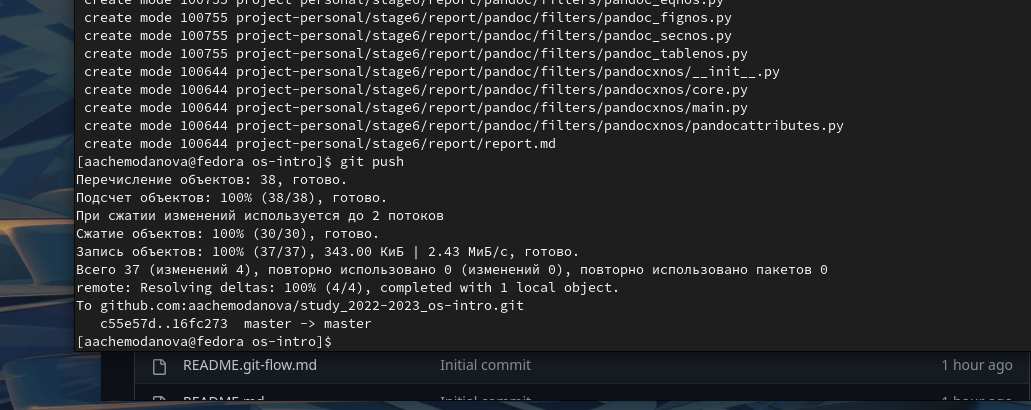


Figure 15: Настройка каталога курса

# 5 Выводы

Мы изучили идеологию и применение средств контроля версий, а также освоили умения по работе с git.

# 6 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.

1. 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.

1. 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.

1. 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.

1. 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.

1. 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.

1. 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.

1. 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

1. 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.

1. 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