Федеральное государственное автономное
Образовательное учреждение высшего образования
Российский Университет Дружбы Народов

Математический университет имени Никольского

Факультет Физико-математических и Естественных наук

Кафедра Прикладной математики и информатики

Отчет по лабораторной работе № 2

“Работа с git”

Выполнил:

Студент группы НПМбв-01-10

Адхамова Луиза

Москва

2024 год

Цель работы:

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

Выполнение работы:

Для начала работы установим git и gh (рис 1.1, 1.2, 1.3):

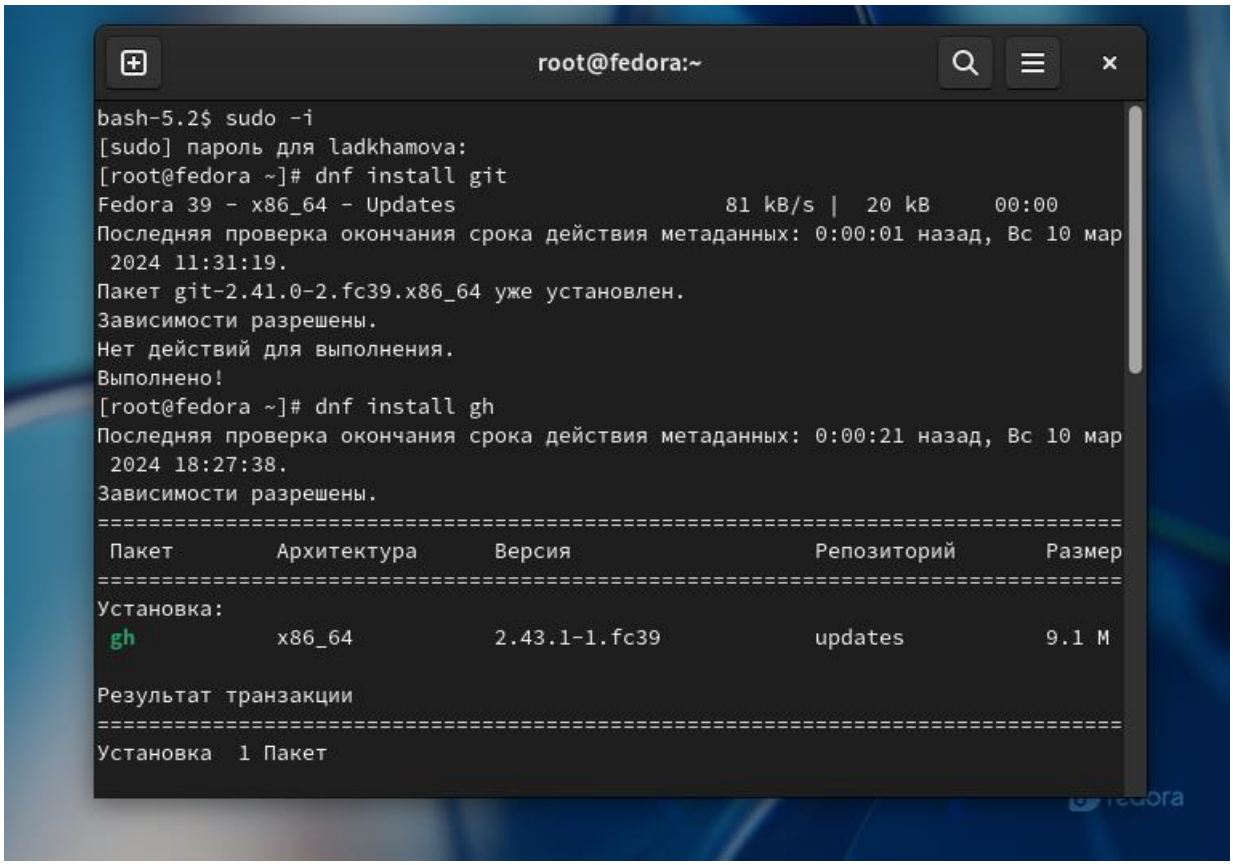

Рисунок 1.1 Установка пакетов git и gh.

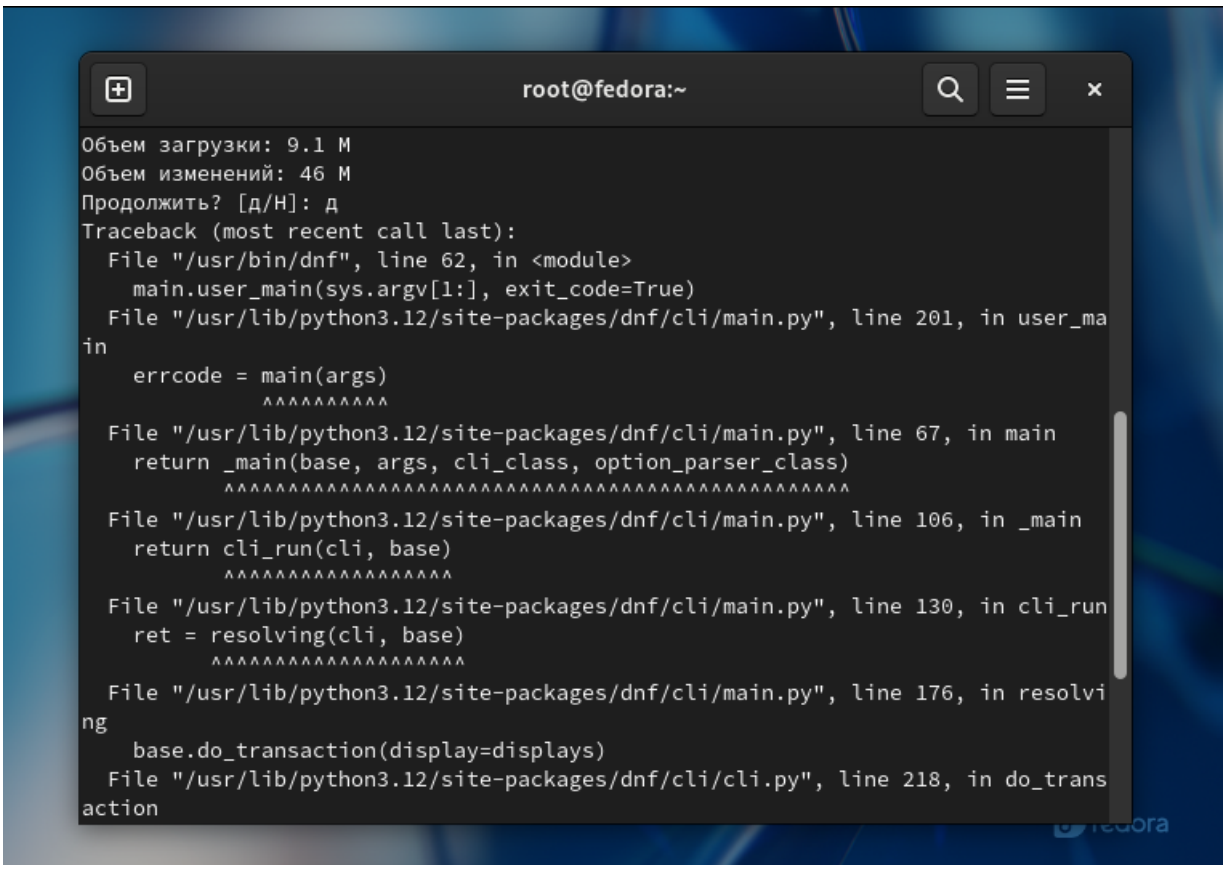

Рисунок 1.2. Продолжение.

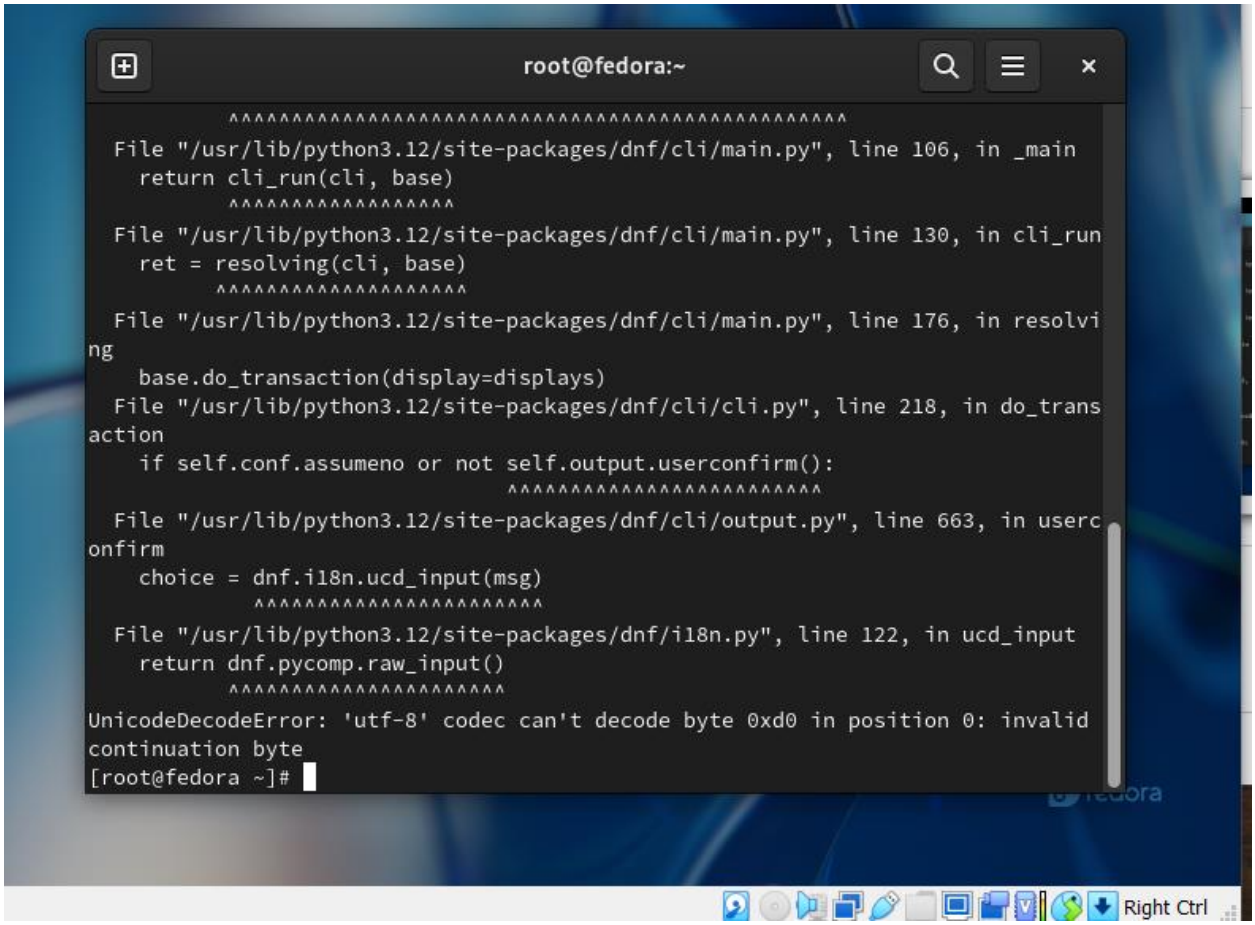

Рисунок 1.3. Продолжение (1).

Зададим имя и email владельца репозитория (рис 2.):

git config --global user.name "Name Surname"
git config --global user.email "work@mail"

И создадим ssh ключ (рис. 2):

по алгоритму *rsa* с ключём размером 4096 бит:

ssh-keygen -t rsa -b 4096

по алгоритму ed25519:

ssh-keygen -t ed25519

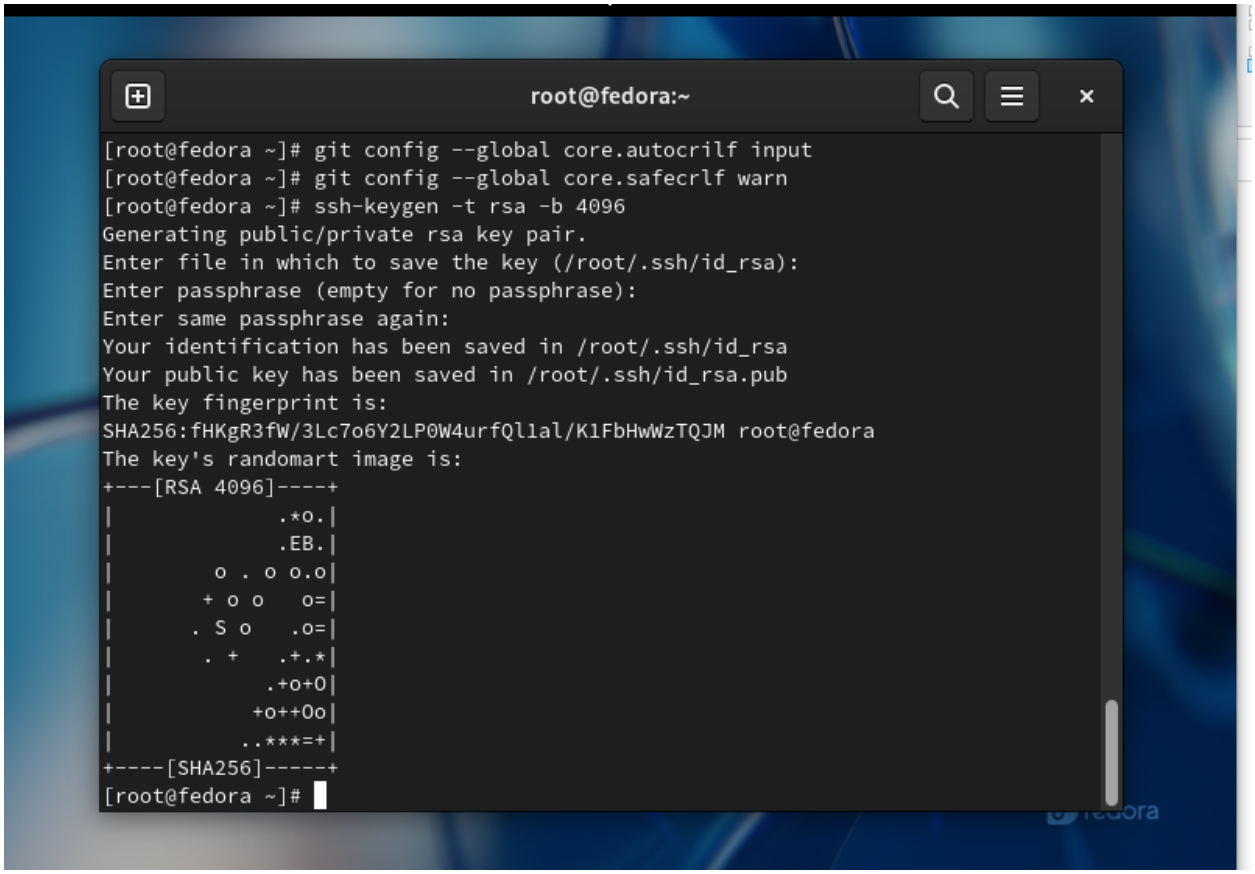

Рисунок 2.1. Ввод имени и email владельца репозитория и создания ssh ключа.

Создадим ключ pgp (рис 3.1, 3.2, 3.3):

Генерируем ключ

gpg --full-generate-key

Из предложенных опций выбираем:

- тип *RSA and RSA*;

- размер 4096;
- выберите срок действия; значение по умолчанию — 0 (срок действия не истекает никогда).

GPG запросит личную информацию, которая сохранится в ключе:

- Имя (не менее 5 символов).
- Адрес электронной почты.

При вводе email убедитесь, что он соответствует адресу, используемому на GitHub.

- Комментарий. Можно ввести что угодно или нажать клавишу ввода, чтобы оставить это поле пустым.

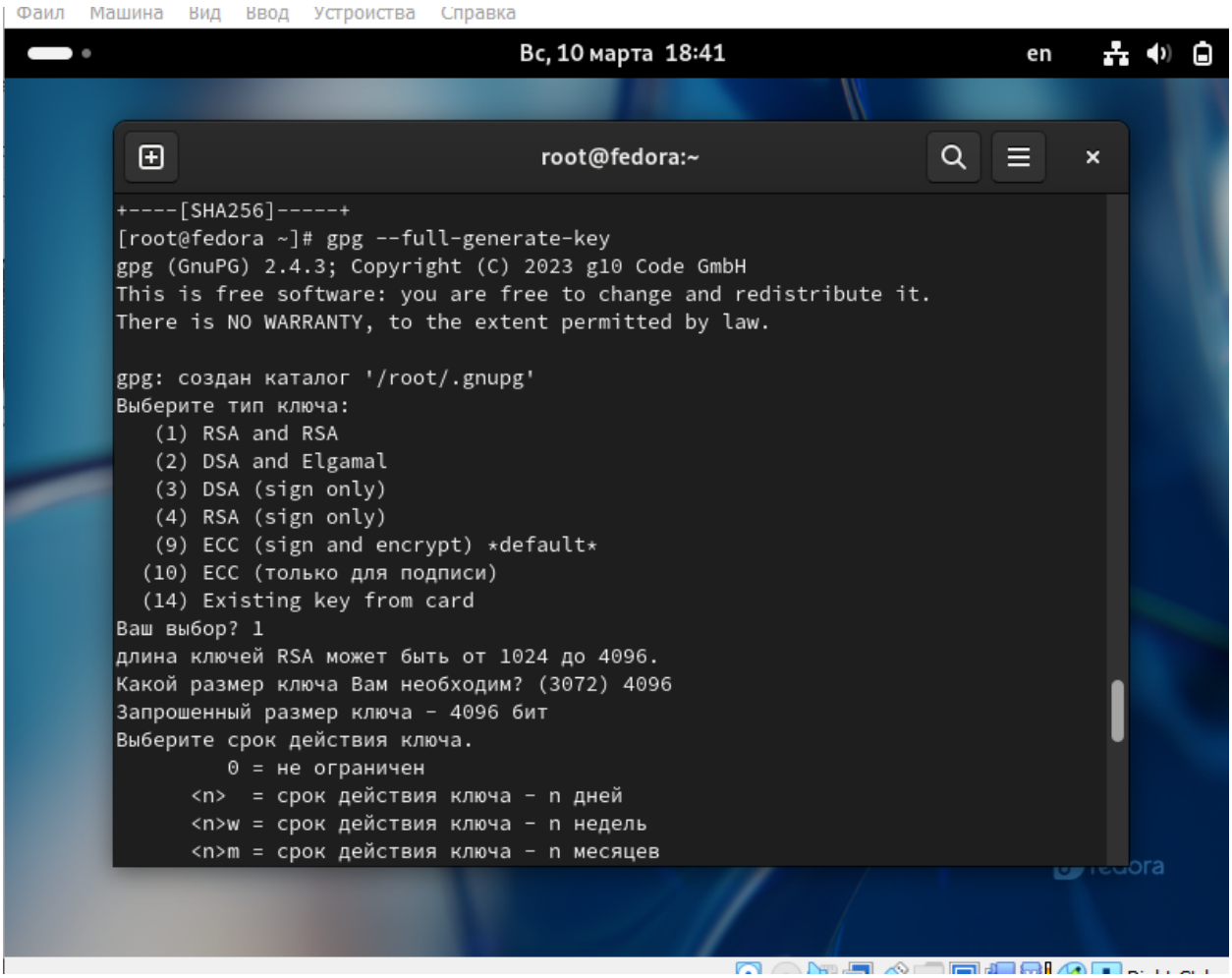

Рисунок 3.1. Создание pgp ключа.

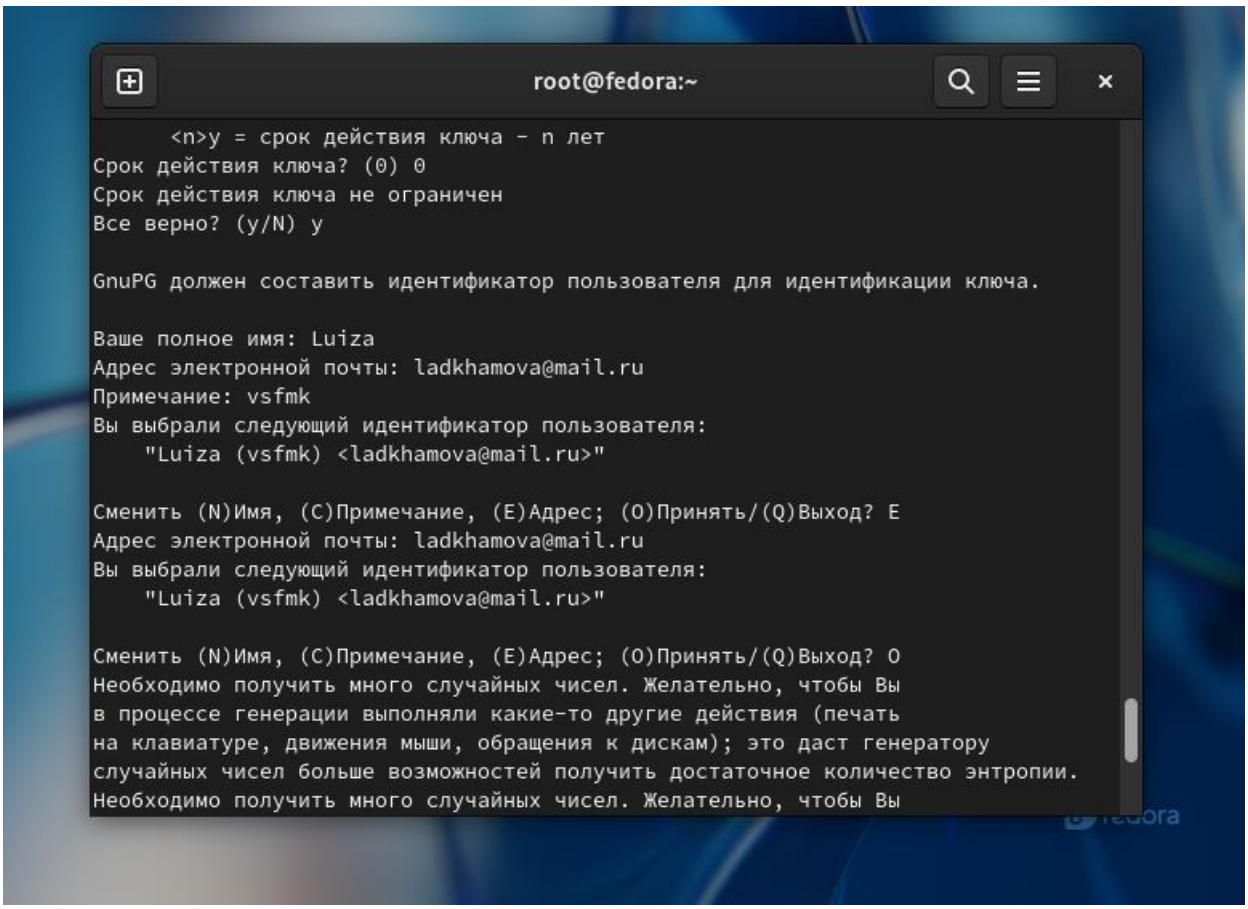

Рисунок 3.2. Продолжение.

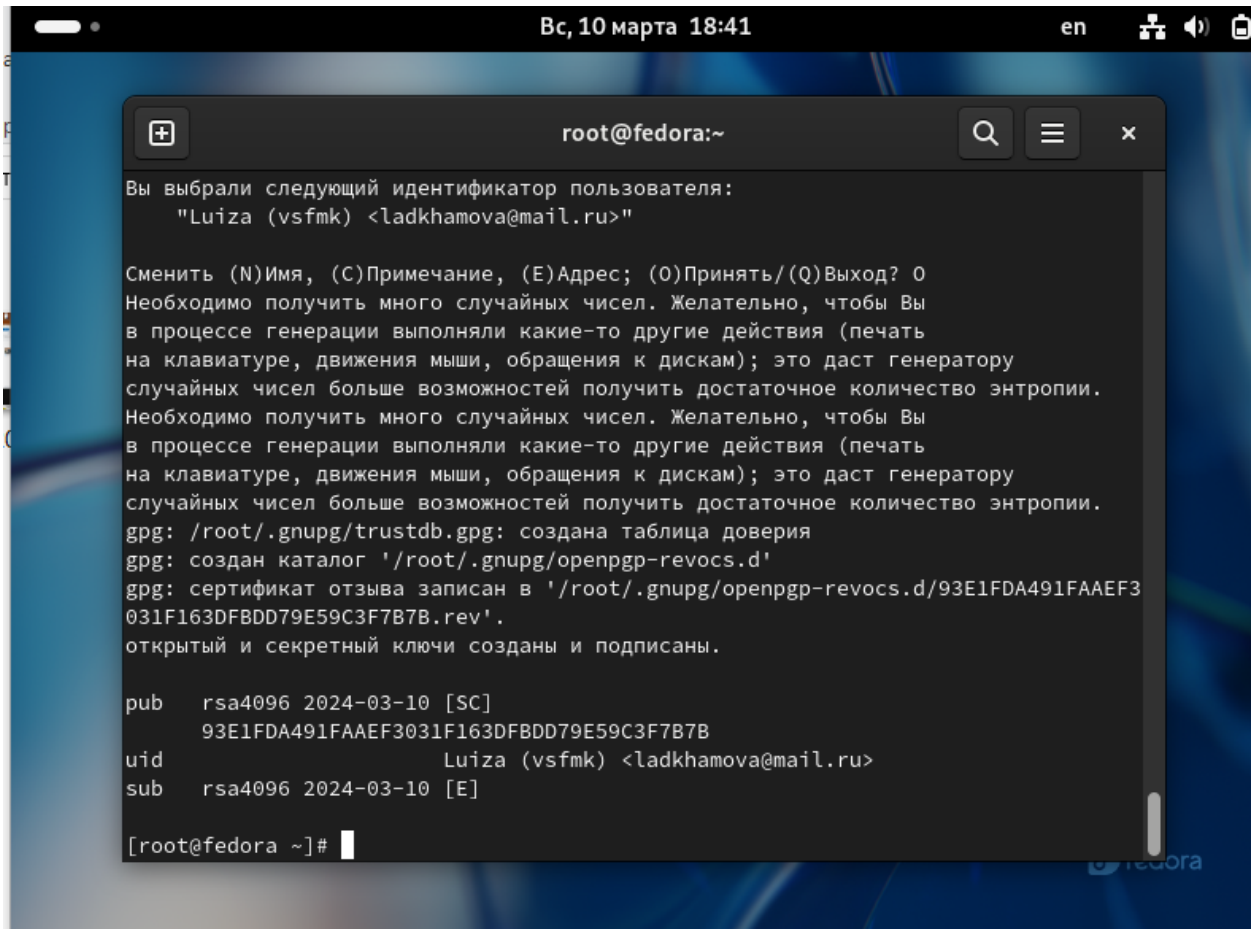

Рисунок 3.3. Продолжение (1).

Регистрироваться на Github не нужно так как уже есть аккаунт (рис 4):

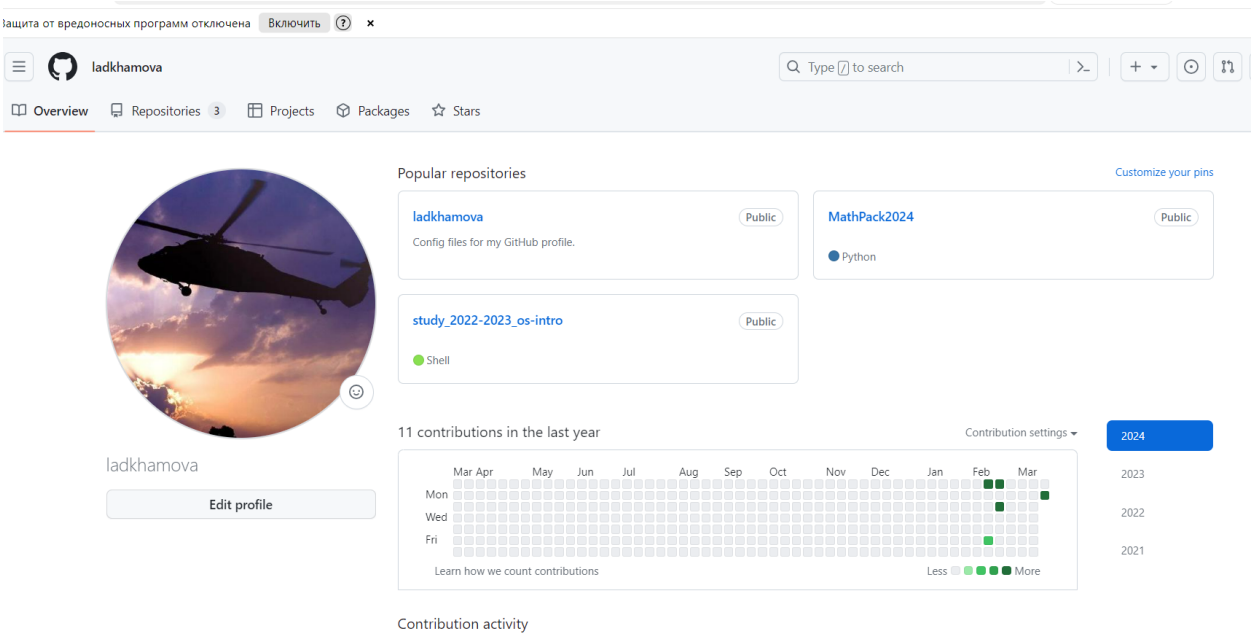

Рисунок 4. Аккаунт на Github.

Ответы на контрольные вопросы:

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

   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.

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

   VCS (Version Control System) – это система управления версиями, которая позволяет отслеживать изменения в файловой системе, сохранять историю версий файлов, и управлять доступом к изменениям.

   Хранилище (repository) – это база данных, в которой хранятся все версии файлов, история изменений, а также метаданные о каждом коммите.

   Commit – это операция, при которой изменения в рабочей копии файлов сохраняются в репозитории. Коммит создает новую версию файла и записывает информацию о внесенных изменениях.

   История (history) – это список всех коммитов, которые были сделаны в репозитории, включая информацию о том, кто, когда и что изменил в файлах.

   Рабочая копия (working copy) – это локальная копия файлов из репозитория, с которой пользователь работает и вносит изменения перед их коммитом. Рабочая копия позволяет просматривать текущее состояние файлов, сравнивать их с последней версией из репозитория и зафиксировать изменения.

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

   VCS. Version Control System (VCS)– система контроля версий. Из названия следует основной кейс применения таких систем – контроль версий систем. VCS сохраняет изменения, которые произошли от одной версии файла к другой. В качестве систем могут быть файлы с кодом программ, скриптов или конфигурационные файлы (например, файлы конфигурации DHCP, файлы зон DNS, настроек iptables или apache).

   В случае с централизованной VCS репозиторий хранится на одном сервере, и все разработчики работают с ним. Очевидное преимущество: простое управление выпуском релизов и вообще ходом развития программы, раз весь код в одном месте. Очевидный недостаток: если с сервером что-то случится, работа всех разработчиков пропадет (даже в случае регулярных бэкапов - пропадет работа всех разработчиков, скажем, за последнюю неделю).

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

При единоличной работе с хранилищем версий (VCS) основными действиями пользователя будут следующие:

Создание репозитория: пользователь создает локальный репозиторий на своем компьютере для хранения всех файлов и изменений.

Добавление файлов: пользователь добавляет файлы в репозиторий с помощью команды "git add" или аналогичной в зависимости от используемой системы контроля версий.

Фиксация изменений: после добавления файлов пользователь фиксирует изменения с помощью команды "git commit" и добавляет комментарий к изменениям.

Ветвление и слияние: при необходимости пользователь создает новые ветки для работы над отдельными функциональностями или исправлениями, а затем объединяет ветки с основной веткой с помощью слияния (merge).

Просмотр истории изменений: пользователь может просматривать историю изменений с помощью команды "git log" и откатываться к предыдущим версиям файлов при необходимости.

Отправка изменений на удаленный сервер: при желании пользователь может загружать свои изменения на удаленный сервер, например на GitHub, с помощью команды "git push".

Получение изменений с удаленного сервера: в случае необходимости пользователь может получить изменения, внесенные другими пользователями, с удаленного сервера с помощью команды "git pull".

Эти действия позволяют пользователю эффективно управлять версиями файлов и сохранять историю изменений при работе с хранилищем версий (VCS) в одиночку.

5. Опишите порядок работы с общим хранилищем VCS.

Создание репозитория: сначала необходимо создать новый репозиторий на сервере, используя специальную команду в командной строке или через веб-интерфейс хостинг-провайдера.

Клонирование репозитория: после того как репозиторий создан, каждый участник проекта должен клонировать его на свой компьютер с помощью команды git clone (для Git) или svn checkout (для SVN).

Работа с файлами: после клонирования участник может начать работу с файлами в своем репозитории. Он может добавлять, изменять и удалять файлы, а затем фиксировать изменения в историю с помощью команд git add (для Git) или svn commit (для SVN).

Обновление репозитория: для того чтобы получить последние изменения из общего репозитория, участник может использовать команду git pull (для Git) или svn update (для SVN).

Отправка изменений: после внесения всех необходимых изменений участник может отправить их в общий репозиторий с помощью команды git push (для Git) или svn commit (для SVN).

Разрешение конфликтов: если возникнут конфликты при отправке изменений, участник должен разрешить их, исправив файлы вручную или используя специальные инструменты для слияния.

Ревью кода: после отправки изменений другие участники проекта могут просмотреть их и оставить комментарии или запросы на изменения. Для этого часто используются специализированные сервисы для код-ревью.

Слияние изменений: когда все изменения приняты и одобрены, они могут быть слиты в общую ветку с помощью команд git merge (для Git) или svn merge (для SVN).

Релизы и теги: для отметки важных моментов в истории проекта, таких как выпуск новой версии, могут быть созданы релизы или теги с помощью команд git tag (для Git) или svn copy (для SVN).

Анализ и управление историей: с помощью различных инструментов участники могут анализировать и управлять историей изменений, изучая коммиты, возвращаясь к предыдущим версиям и т.д.

6. Каковы основные задачи, решаемые инструментальным средством git?

   Git — это система управления версиями. У **Git** две **основных задачи**: первая — хранить информацию о 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.

   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:

   1. добавить все изменённые и/или созданные файлы и/или каталоги:
   2. `git add .`

3. добавить конкретные изменённые и/или созданные файлы и/или каталоги:
4. `git add имена_файлов`
5. удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории):
6. `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:

7. удаление локальной уже слитой с основным деревом ветки:
8. `git branch -d имя_ветки`
9. принудительное удаление локальной ветки:
10. `git branch -D имя_ветки`
11. удаление ветки с центрального репозитория:

```
git push origin :имя_ветки
```

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

Пример использования локального репозитория:

Создание нового репозитория:

csharp

git init

Добавление файлов в индекс:

$ git add <file>

Создание коммита:

$ git commit -m "Initial commit"

Пример использования удаленного репозитория:

Клонирование удаленного репозитория:

$ git clone <remote_url>

Добавление удаленного репозитория:

$ git remote add origin <remote_url>

Отправка изменений на удаленный репозиторий:

$ git push origin master

Таким образом, локальный репозиторий используется для хранения и работы с изменениями в коде, а удаленный репозиторий используется для совместной работы с другими разработчиками и сохранения истории изменений.

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

При помощи веток в VCS можно:

- Реализовать фичу, не мешая остальным.
- Проводить модерацию (кодревью) нового кода перед непосредственным добавлением в кодовую базу.
- Отвлечься от реализации фичи и починить баг в другом месте.
- Вовсе отложить начатую фичу до лучших времен.
- Получить запрос на доработку старой версии программы от заказчика и поддерживать далее несколько версий ПО.
- Поэкспериментировать с кодом без страха сломать билд.
- Организовать процесс поэтапного выпуска программы (разработка - тестирование - релиз), не блокируя разработку следующей версии.
- Организовать работу с open source сообществом или подрядчиком.
- Запилить постоянную автоматическую сборку с рабочей ветки с прогоном тестов и ручную авторизованную сборку релиза с релиз-ветки.

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

Для игнорирования некоторых файлов при коммите в репозиторий можно использовать файл .gitignore. В этом файле можно указать шаблоны для игнорирования определенных

файлов или директорий. Например, чтобы игнорировать все файлы с расширением .log, нужно добавить в файл .gitignore следующую строку:

*.log

Также можно указать конкретные файлы или директории, которые необходимо игнорировать. Например, чтобы игнорировать файл secret.txt, добавьте строку:

secret.txt

Игнорирование файлов при коммите полезно в случае, если некоторые файлы содержат конфиденциальную информацию, временные файлы или сгенерированные файлы, которые не должны попасть в репозиторий. Также это позволяет поддерживать репозиторий чистым и удобным для работы.