Шаблон отчёта по лабораторной работе №5

настройка рабочей среды

Кализая Арсе Кхари Жекка

Содержание

# 1 Цель работы

Создать и соединять репозиторий с утилитами, используя chezmoi.

# 2 Задание

установить утилиты, настроить интерфейс и подключить репозиторий к своей системе.

# 3 Теоретическое введение

## 3.1 Менеджер паролей pass

* Менеджер паролей pass — программа, сделанная в рамках идеологии Unix.
* Также носит название стандартного менеджера паролей для Unix (The standard Unix password manager).

### 3.1.1 Основные свойства

* Данные хранятся в файловой системе в виде каталогов и файлов.
* Файлы шифруются с помощью GPG-ключа.

### 3.1.2 Структура базы паролей

* Структура базы может быть произвольной, если Вы собираетесь использовать её напрямую, без промежуточного программного обеспечения. Тогда семантику структуры базы данных Вы держите в своей голове.
* Если же необходимо использовать дополнительное программное обеспечение, необходимо семантику заложить в структуру базы паролей.

#### 3.1.2.1 Семантическая структура базы паролей

* Рассмотрим пользователя user в домене example.com, порт 22.
* Отсутствие имени пользователя или порта в имени файла означает, что любое имя пользователя и порт будут совпадать:
* example.com.pgp
* Соответствующее имя пользователя может быть именем файла внутри каталога, имя которого совпадает с хостом. Это полезно, если в базе есть пароли для нескольких пользователей на одном хосте:
* example.com/user.pgp
* Имя пользователя также может быть записано в виде префикса, отделенного от хоста знаком @:
* user@example.com.pgp
* Соответствующий порт может быть указан после хоста, отделённый двоеточием (:):
* example.com:22.pgp  
   example.com:22/user.pgp  
   user@example.com:22.pgp
* Эти все записи могут быть расположены в произвольных каталогах, задающих Вашу собственную иерархию.

### 3.1.3 Реализации

#### 3.1.3.1 Утилиты командной строки

* На данный момент существует 2 основных реализации:
  + pass — классическая реализация в виде shell-скриптов (https://www.passwordstore.org/);
  + gopass — реализация на go с дополнительными интегрированными функциями (https://www.gopass.pw/).
* Дальше в тексте будет использоваться программа pass, но всё то же самое можно сделать с помощью программы gopass.

#### 3.1.3.2 Графические интерфейсы

1. qtpass
   * qtpass — может работать как графический интерфейс к pass, так и как самостоятельная программа. В настройках можно переключаться между использованием pass и gnupg.
2. gopass-ui
   * gopass-ui — интерфейс к gopass.
3. webpass
   * Репозиторий: https://github.com/emersion/webpass
   * Веб-интерфейс к pass.
   * Написано на golang.

#### 3.1.3.3 приложения для Android

1. Password Store
   * URL: https://play.google.com/store/apps/details?id=dev.msfjarvis.aps
   * Репозиторий с кодом: https://github.com/android-password-store/Android-Password-Store
   * Документация: https://android-password-store.github.io/docs/
   * Для синхронизации с git необходимо импортировать ssh-ключи.
   * Поддерживает разблокировку по биометрическим данным.
   * Для работы требует наличия OpenKeychain: Easy PGP.
2. OpenKeychain: Easy PGP
   * URL: https://play.google.com/store/apps/details?id=org.sufficientlysecure.keychain
   * Операции с ключами pgp.
   * Необходимо будет импортировать pgp-ключи.
   * Не поддерживает разблокировку по биометрическим данным. Необходимо набирать пароль ключа.

#### 3.1.3.4 Пакеты для Emacs

1. pass
   * Основной режим для управления хранилищем и редактирования записей.
   * Emacs. Пакет pass
   * Репозиторий: https://github.com/NicolasPetton/pass
   * Позволяет редактировать базу данных паролей.
   * Запуск
   * M-x pass
2. helm-pass
   * Интерфейс helm для pass.
   * Репозиторий: https://github.com/emacs-helm/helm-pass
   * Запуск
   * M-x helm-pass
   * Выдаёт в минибуфере список записей из базы паролей. При нажатии Enter копирует пароль в буфер.
3. ivy-pass
   * Интерфейс ivy для pass.
   * Репозиторий: https://github.com/ecraven/ivy-pass

## 3.2 Управление файлами конфигурации

* Использование chezmoi для управления файлами конфигурации домашнего каталога пользователя.

### 3.2.1 Обшая информация

* Сайт: https://www.chezmoi.io/
* Репозиторий: https://github.com/twpayne/chezmoi

### 3.2.2 Конфигурация *chezmoi*

#### 3.2.2.1 Рабочие файлы

* Состояние файлов конфигурации сохраняется в каталоге ~/.local/share/chezmoi
* Он является клоном вашего репозитория dotfiles.
* Файл конфигурации ~/.config/chezmoi/chezmoi.toml (можно использовать также JSON или YAML) специфичен для локальной машины.
* Файлы, содержимое которых одинаково на всех ваших машинах, дословно копируются из исходного каталога.
* Файлы, которые варьируются от машины к машине, выполняются как шаблоны, обычно с использованием данных из файла конфигурации локальной машины для настройки конечного содержимого, специфичного для локальной машины.
* При запуске chezmoi apply

вычисляется желаемое содержимое и разрешения для каждого файла, а затем вносит необходимые изменения, чтобы ваши файлы соответствовали этому состоянию. - По умолчанию chezmoi изменяет файлы только в рабочей копии.

#### 3.2.2.2 Автоматически создавать файл конфигурации на новой машине

* При выполнении chezmoi init также может автоматически создать файл конфигурации, если он еще не существует.
* Если ваш репозиторий содержит файл с именем .chezmoi.$FORMAT.tmpl, где $FORMAT есть один из поддерживаемых форматов файла конфигурации (json, toml, или yaml), то chezmoi init выполнит этот шаблон для создания исходного файла конфигурации.
* Например, пусть ~/.local/share/chezmoi/.chezmoi.toml.tmpl выглядит так:
* {{- $email := promptStringOnce . "email" "Email address" -}}  
    
   [data]  
   email = {{ $email | quote }}
  + При выполнении chezmoi init будет создан конфигурационный файл ~/.config/chezmoi/chezmoi.toml.
  + promptStringOnce — это специальная функция, которая запрашивает у пользователя значение, если оно еще не установлено в разделе data конфигурационного файла.
* Чтобы протестировать этот шаблон, используйте chezmoi execute-template с флагами –init и –promptString, например:
* chezmoi execute-template --init --promptString email=me@home.org < ~/.local/share/chezmoi/.chezmoi.toml.tmpl

#### 3.2.2.3 Пересоздание файл конфигурации

* Если вы измените шаблон файла конфигурации, chezmoi предупредит вас, если ваш текущий файл конфигурации не был сгенерирован из этого шаблона.
* Вы можете повторно сгенерировать файл конфигурации, запустив:
* chezmoi init

### 3.2.3 Шаблоны

#### 3.2.3.1 Общая информация

* Шаблоны используются для изменения содержимого файла в зависимости от среды.
* Используется синтаксис шаблонов Go.
* Файл интерпретируется как шаблон, если выполняется одно из следующих условий:
  + имя файла имеет суффикс .tmpl;
  + файл находится в каталоге .chezmoitemplates.

#### 3.2.3.2 Данные шаблона

* Полный список переменных шаблона:
* chezmoi data
* Источники переменных:
  + файлы .chezmoi, например, .chezmoi.os;
  + файлы конфигурации .chezmoidata.$FORMAT. Форматы (json, jsonc, toml, yaml) читаются в алфавитном порядке;
  + раздел data конфигурационного файла.

#### 3.2.3.3 Способы создания файла шаблона

* При первом добавлении файла передайте аргумент –template:
* chezmoi add --template ~/.zshrc
* Если файл уже контролируется chezmoi, но не является шаблоном, можно сделать его шаблоном:
* chezmoi chattr +template ~/.zshrc
* Можно создать шаблон вручную в исходном каталоге, присвоив ему расширение .tmpl:
* chezmoi cd  
   $EDITOR dot\_zshrc.tmpl
* Шаблоны в каталоге .chezmoitemplates должны создаваться вручную:
* chezmoi cd  
   mkdir -p .chezmoitemplates  
   cd .chezmoitemplates  
   $EDITOR mytemplate

#### 3.2.3.4 Редактирование файла шаблона

* Используйте chezmoi edit:
* chezmoi edit ~/.zshrc
* Чтобы сделанные вами изменения сразу же применялись после выхода из редактора, используйте опцию –apply:
* chezmoi edit --apply ~/.zshrc

#### 3.2.3.5 Тестирование шаблонов

* Тестирование с помощью команды chezmoi execute-template.
* Тестирование небольших фрагментов шаблонов:
* chezmoi execute-template '{{ .chezmoi.hostname }}'
* Тестирование целых файлов:
* chezmoi cd  
   chezmoi execute-template < dot\_zshrc.tmpl

#### 3.2.3.6 Синтаксис шаблона

* Действия шаблона записываются внутри двойных фигурных скобок, {{ }}.
* Действия могут быть переменными, конвейерами или операторами управления.
* Текст вне действий копируется буквально.
* Переменные записываются буквально:
* {{ .chezmoi.hostname }}
* Условные выражения могут быть записаны с использованием if, else if, else, end:
* {{ if eq .chezmoi.os "darwin" }}

darwin

{{ else if eq .chezmoi.os "linux" }}

linux

{{ else }}

other operating system

{{ end }}

1. Удаление пробелов

* Для удаления проблем в шаблоне разместите знак минус и пробела рядом со скобками:
* HOSTNAME={{- .chezmoi.hostname }}
* В результате получим:
* HOSTNAME=myhostname

1. Отладка шаблона

* Используется подкоманда execute-template:
* chezmoi execute-template '{{ .chezmoi.os }}/{{ .chezmoi.arch }}'
* Интерпретируются любые данные, поступающие со стандартного ввода или в конце команды.
* Можно передать содержимое файла этой команде:
* cat foo.txt | chezmoi execute-template

1. Логические операции

* Возможно выполнение логических операций.
* Если имя хоста машины равно work-laptop, текст между if и end будет включён в результат:
* # common config  
   export EDITOR=vi  
    
   # machine-specific configuration  
   {{- if eq .chezmoi.hostname "work-laptop" }}  
   # this will only be included in ~/.bashrc on work-laptop  
   {{- end }}
  1. Логические функции
     + eq: возвращает true, если первый аргумент равен любому из остальных аргументов, может принимать несколько аргументов;
     + not: возвращает логическое отрицание своего единственного аргумента;
     + and: возвращает логическое И своих аргументов, может принимать несколько аргументов;
     + or: возвращает логическое ИЛИ своих аргументов, может принимать несколько аргументов.
  2. Целочисленные функции
     + len: возвращает целочисленную длину своего аргумента;
     + eq: возвращает логическую истину arg1 == arg2;
     + ne: возвращает логическое значение arg1 != arg2;
     + lt: возвращает логическую истину arg1 < arg2;
     + le: возвращает логическую истину arg1 <= arg2;
     + gt: возвращает логическую истину arg1 > arg2;
     + ge: возвращает логическую истину arg1 >= arg2.

### 3.2.4 Переменные шаблона

* Чтобы просмотреть переменные, доступные в вашей системе, выполните:
* chezmoi data
* Чтобы получить доступ к переменной chezmoi.kernel.osrelease в шаблоне, используйте:
* {{ .chezmoi.kernel.osrelease }}

Более подробно про Unix см. в [1–4].

# 4 Выполнение лабораторной работы

## 4.1 установка

я начал эту лабораторную работу, устанавливая pass и gopass (рис. 1) (рис. 2).

dnf install pass pass-otp  
 dnf install gopass

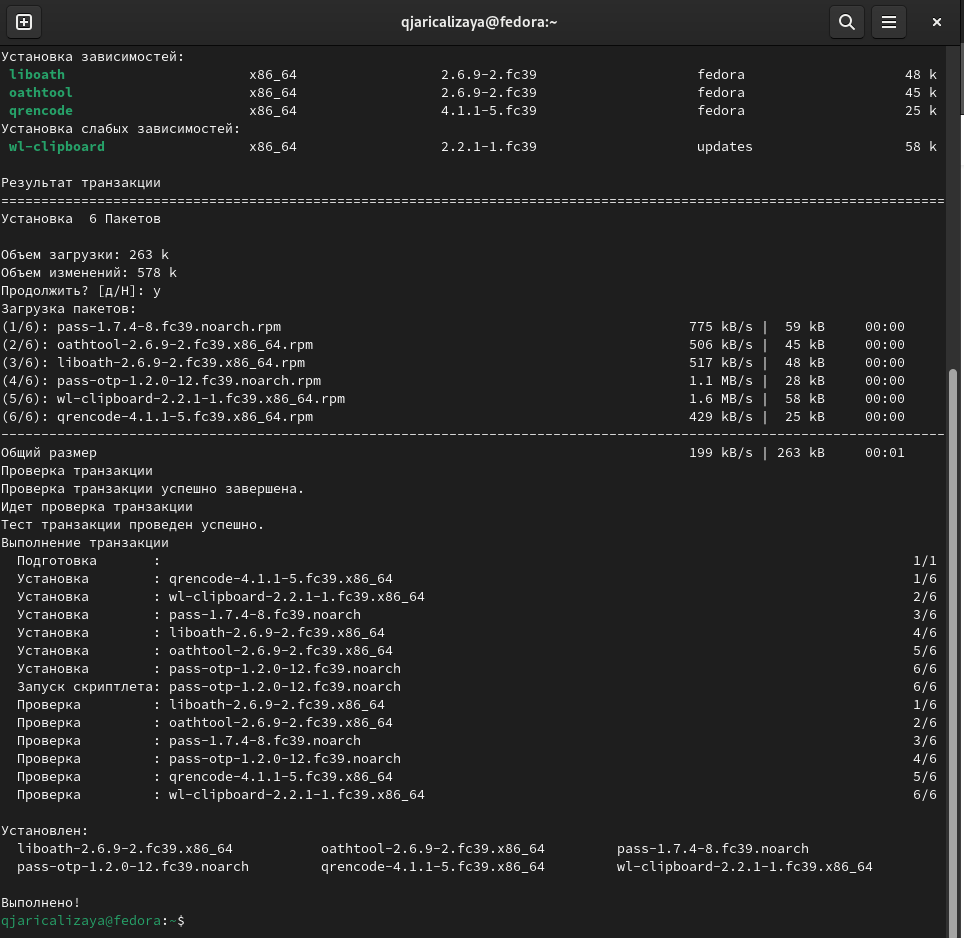


Рис. 1: Установка pass

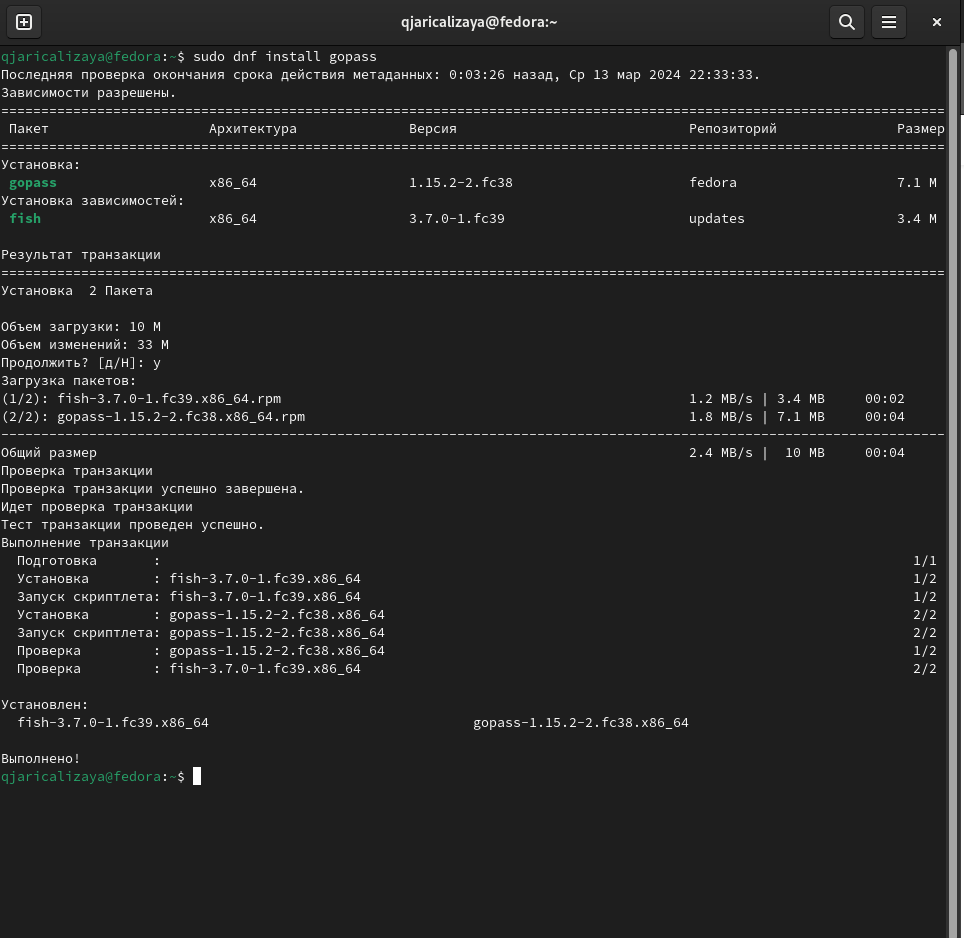


Рис. 2: установка gopass

## 4.2 настройка

Потом я настроил ключ gpg (рис. 3).

gpg --list-secret-keys

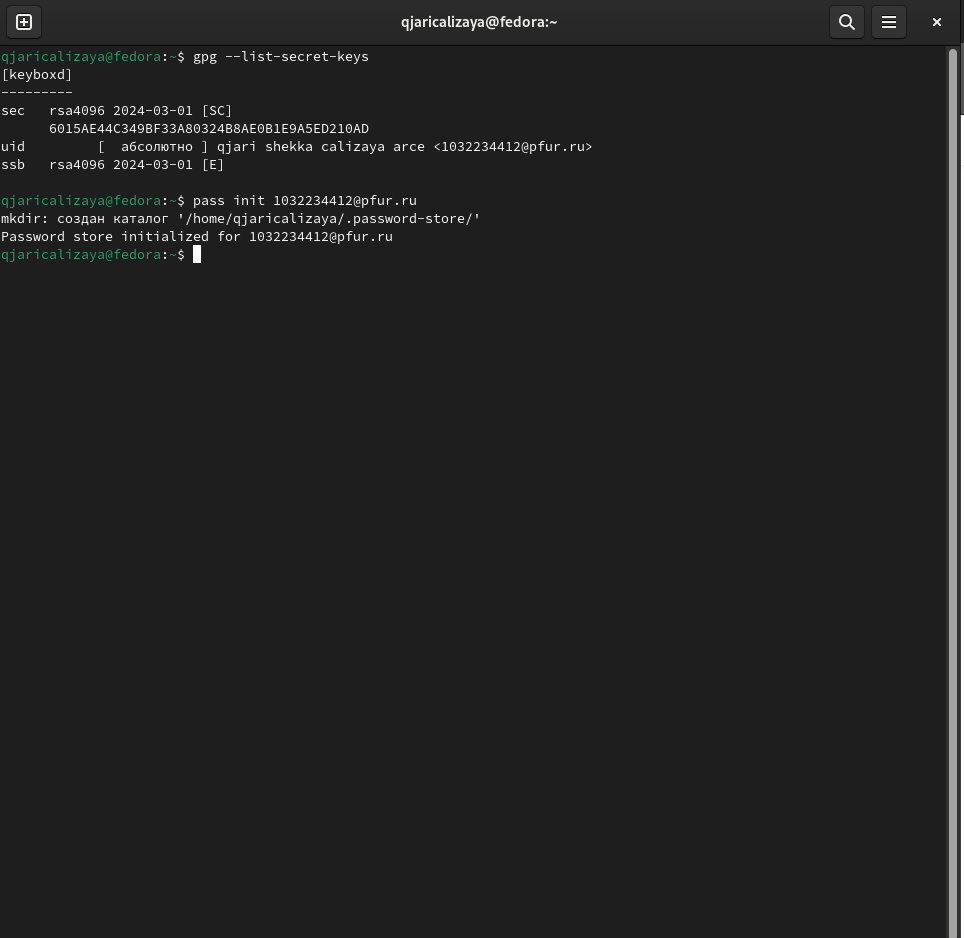


Рис. 3: настройка ключа

У меня уже ключ, поэтому я не создал новой ключ.

Затем я инициализироовал хранилища (рис. 4).

pass init <gpg-id or email>  
  
 pass git init

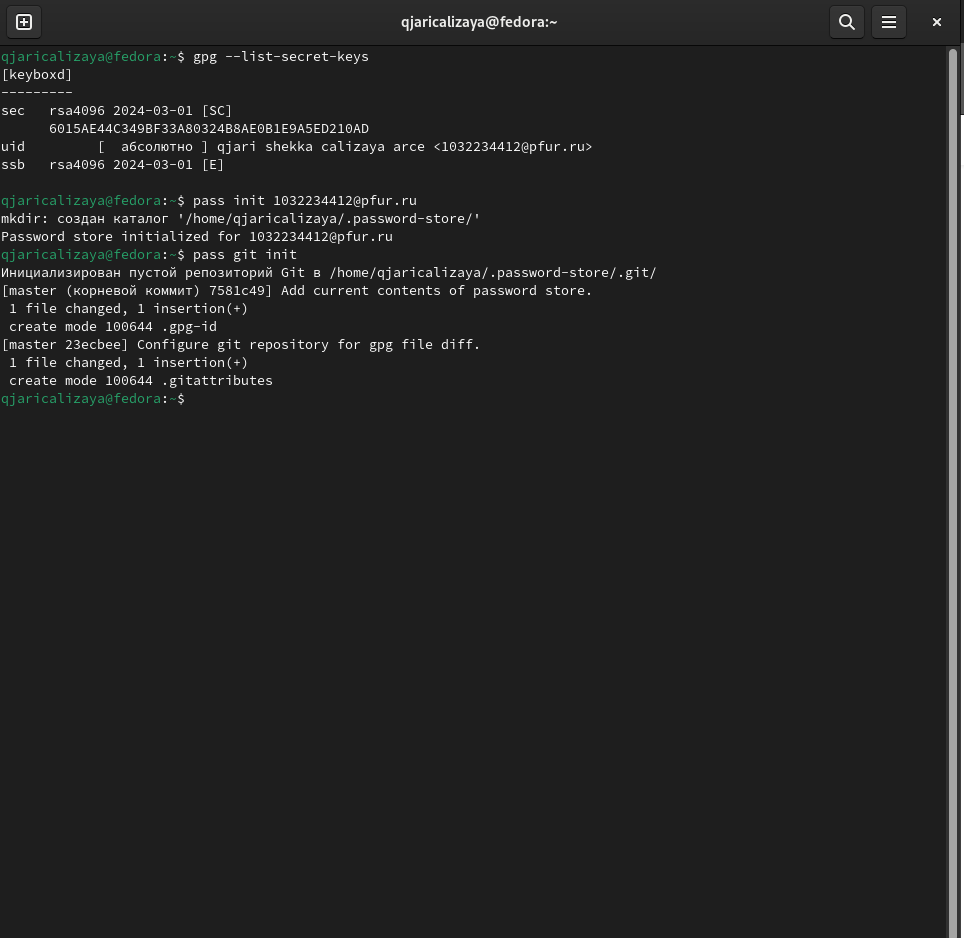


Рис. 4: инициализация

Дальше я задал адрес репозитория на хостинге. (рис. 5).

pass git remote add origin git@github.com:<git\_username>/<git\_repo>.git

Потом я начинал синхронизацию. (рис. 5).

pass git pull  
 pass git push

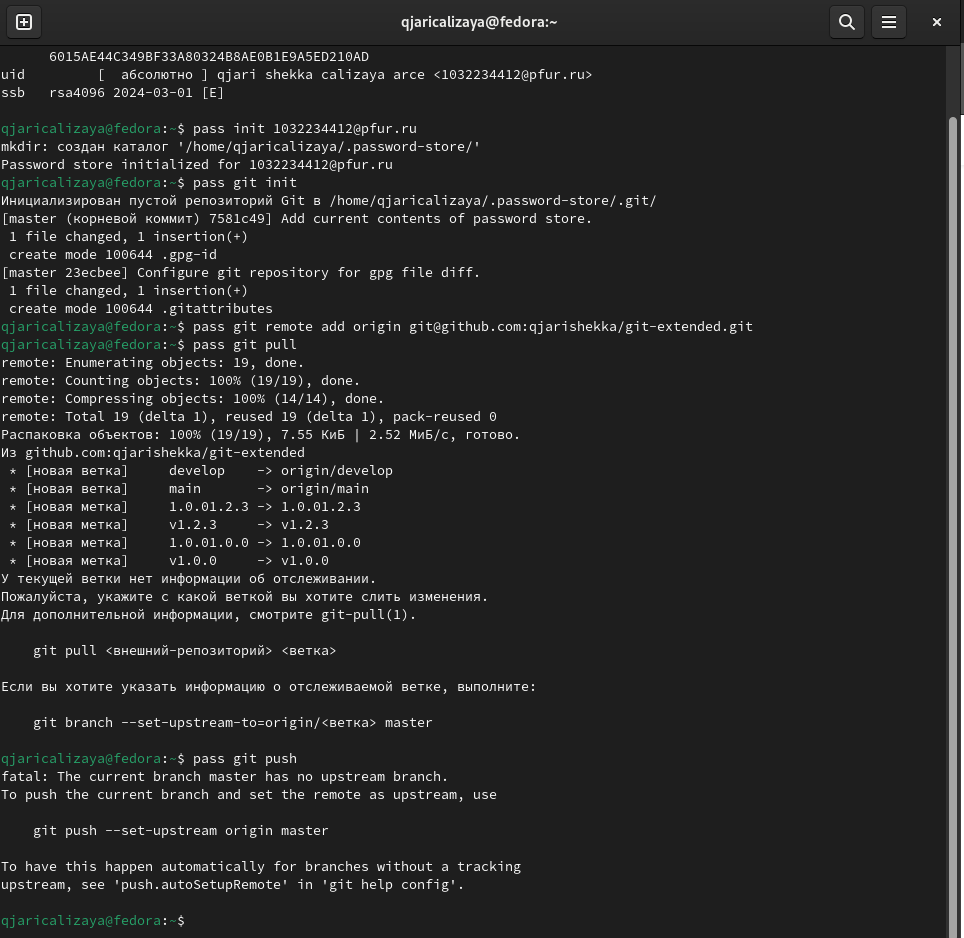


Рис. 5: синхронизация

Потом я сделал изменение непосредственно на файловой системе и закоммитить вручную изменения. (рис. 6).

cd ~/.password-store/  
 git add .  
 git commit -am 'edit manually'  
 git push

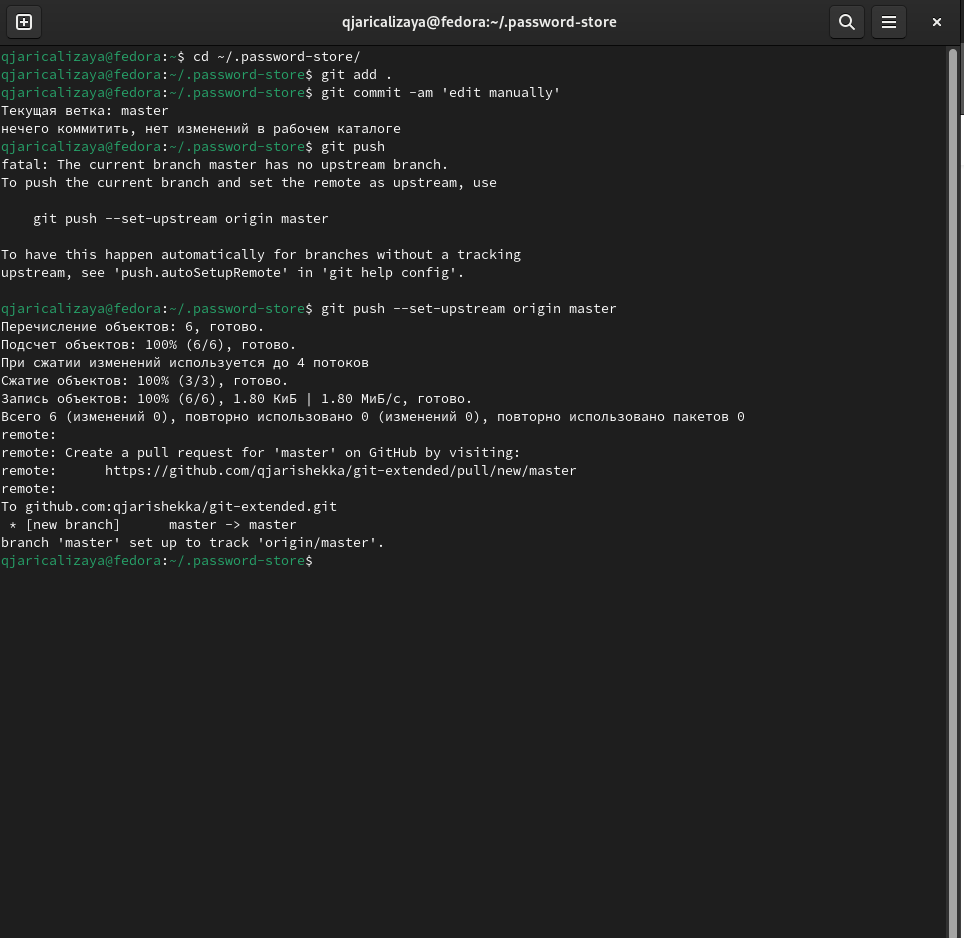


Рис. 6: загрузка на сервер

Затем я проверил статус синхронизации (рис. 7).

pass git status

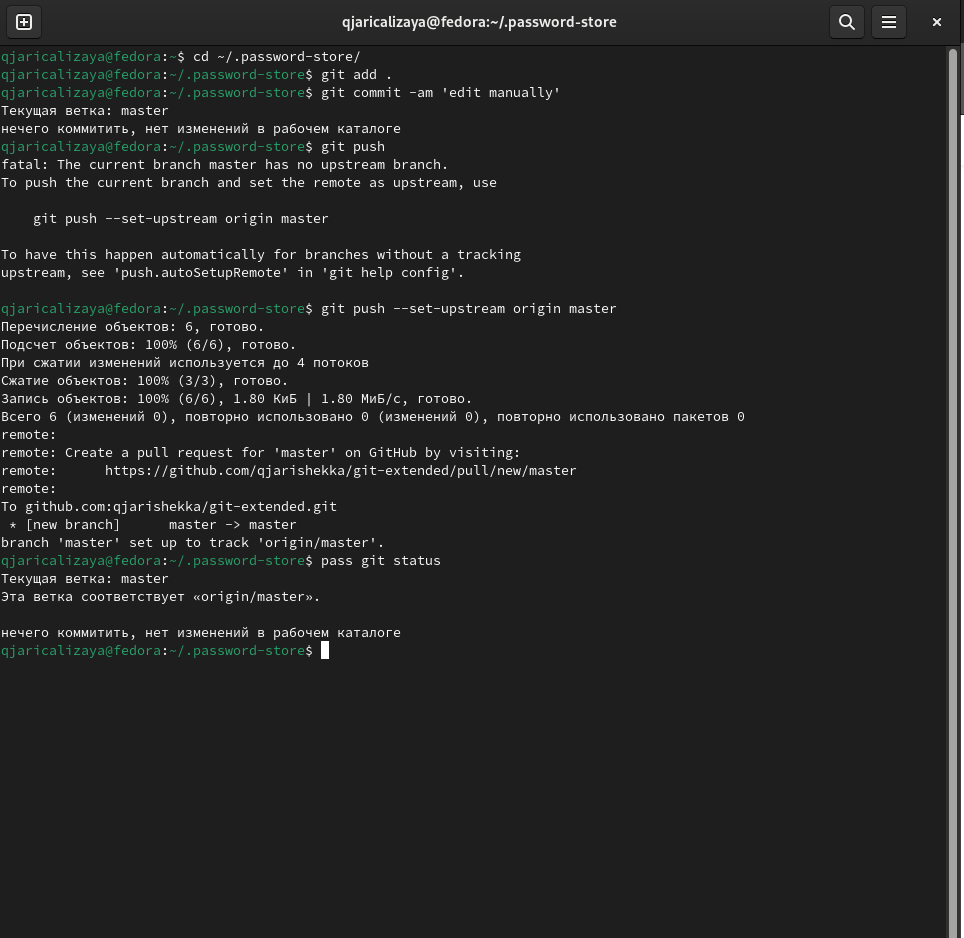


Рис. 7: Status

## 4.3 Настройка интерфейса с броузером

Во-первых я установил add-on в броузере (рис. 8).

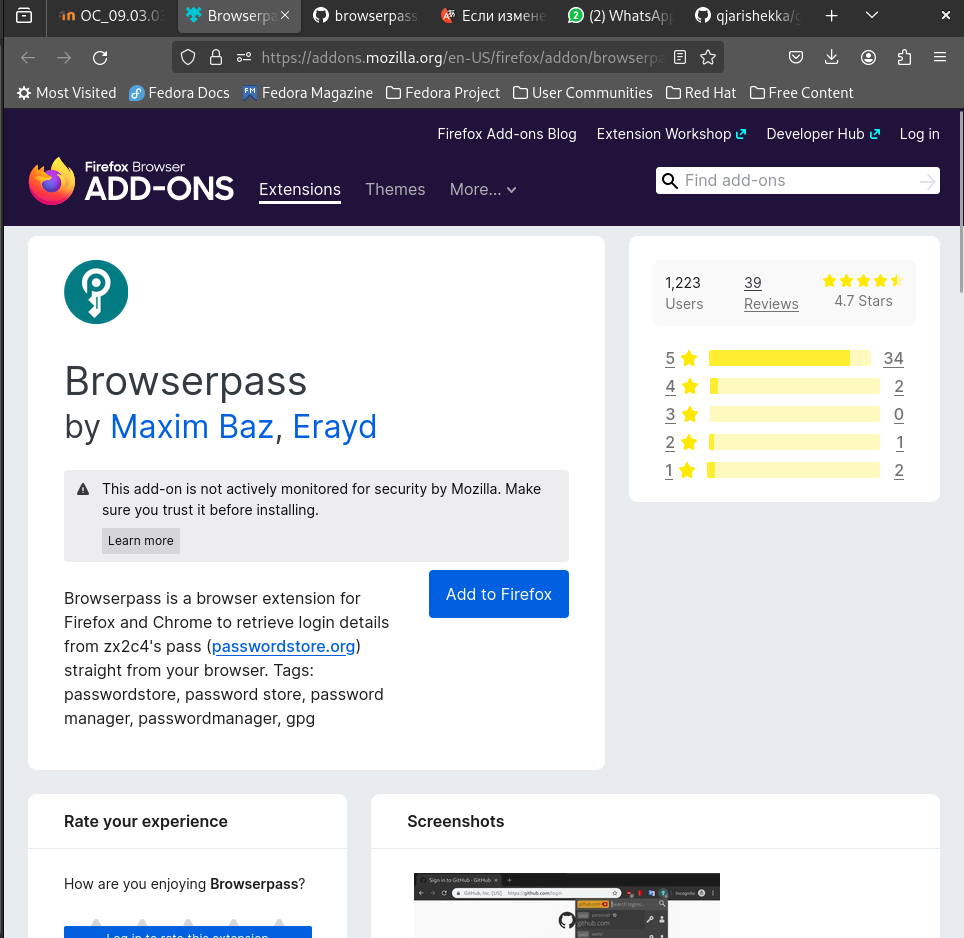


Рис. 8: Настройка браузера

Во-вторых я установил browser pass. (рис. 9) и (рис. 10) .

dnf copr enable maximbaz/browserpass  
 dnf install browserpass

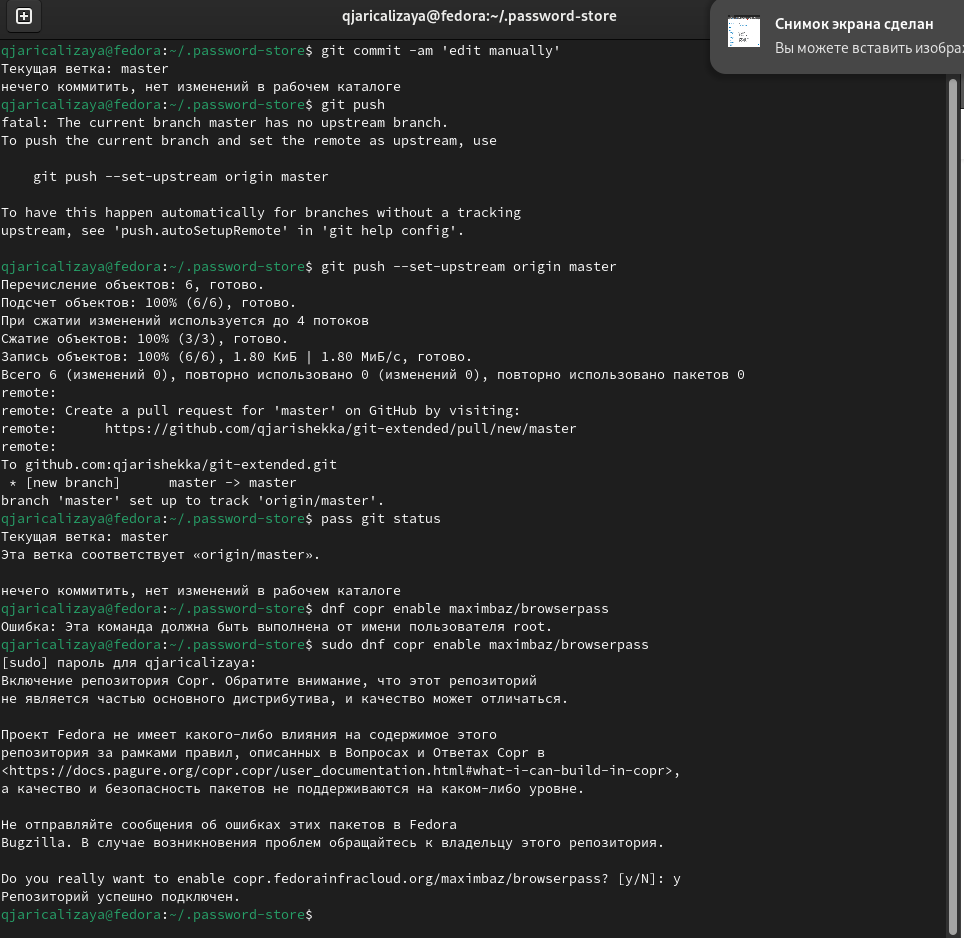


Рис. 9: установка browserpass

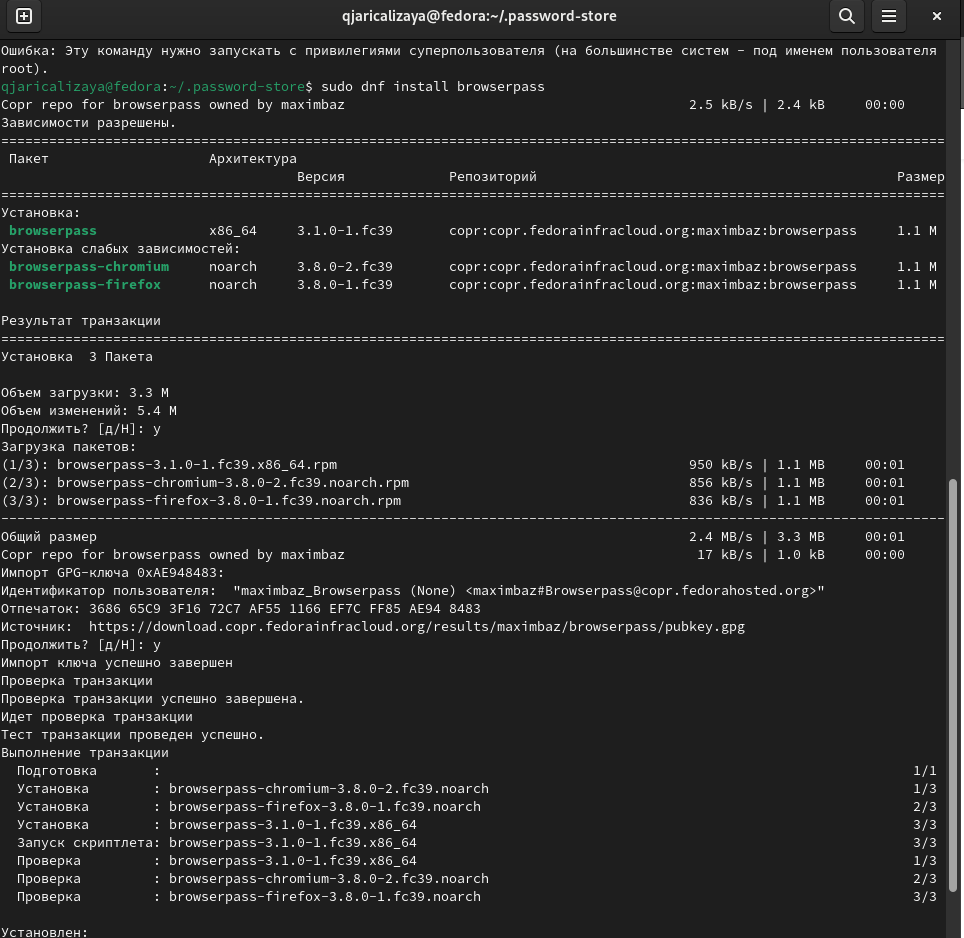


Рис. 10: установка browserpass

## 4.4 Сохранение пароля

я добавил новый пароль (рис. 11) и сохранил его в каталоге password.

pass insert password/password

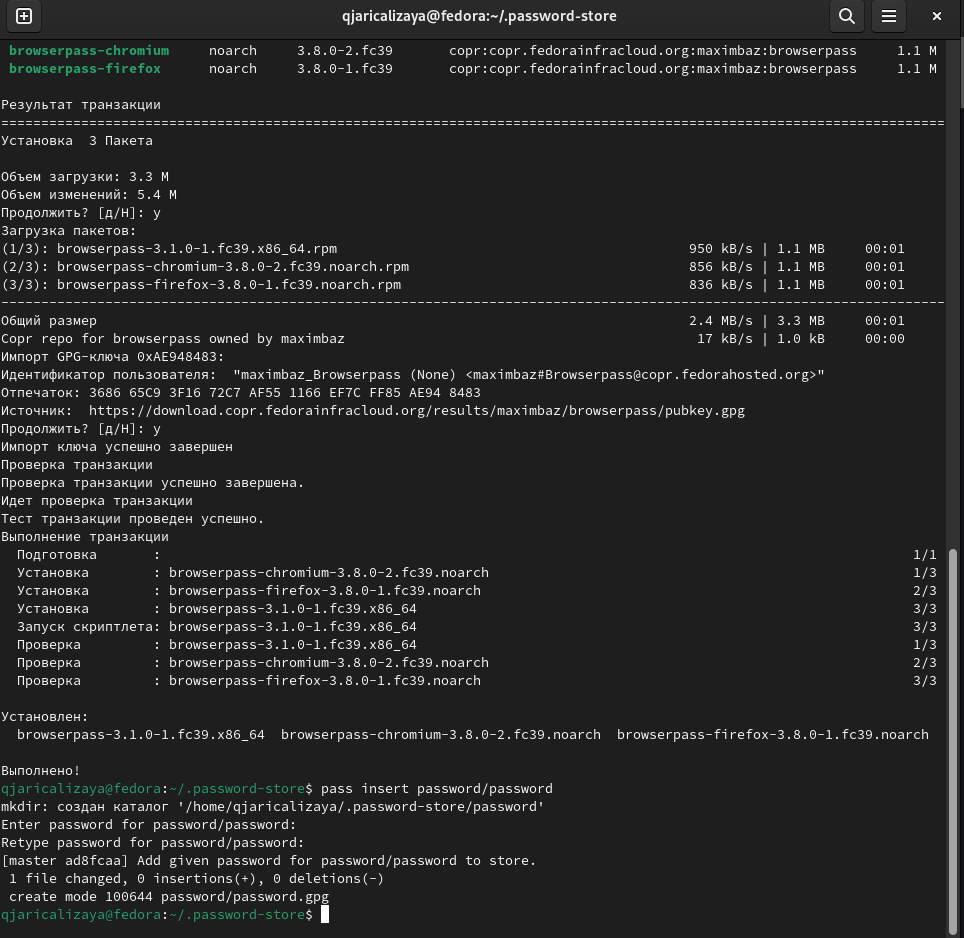


Рис. 11: пароль

Потом я отобразил пароль (рис. 12).

pass password/password

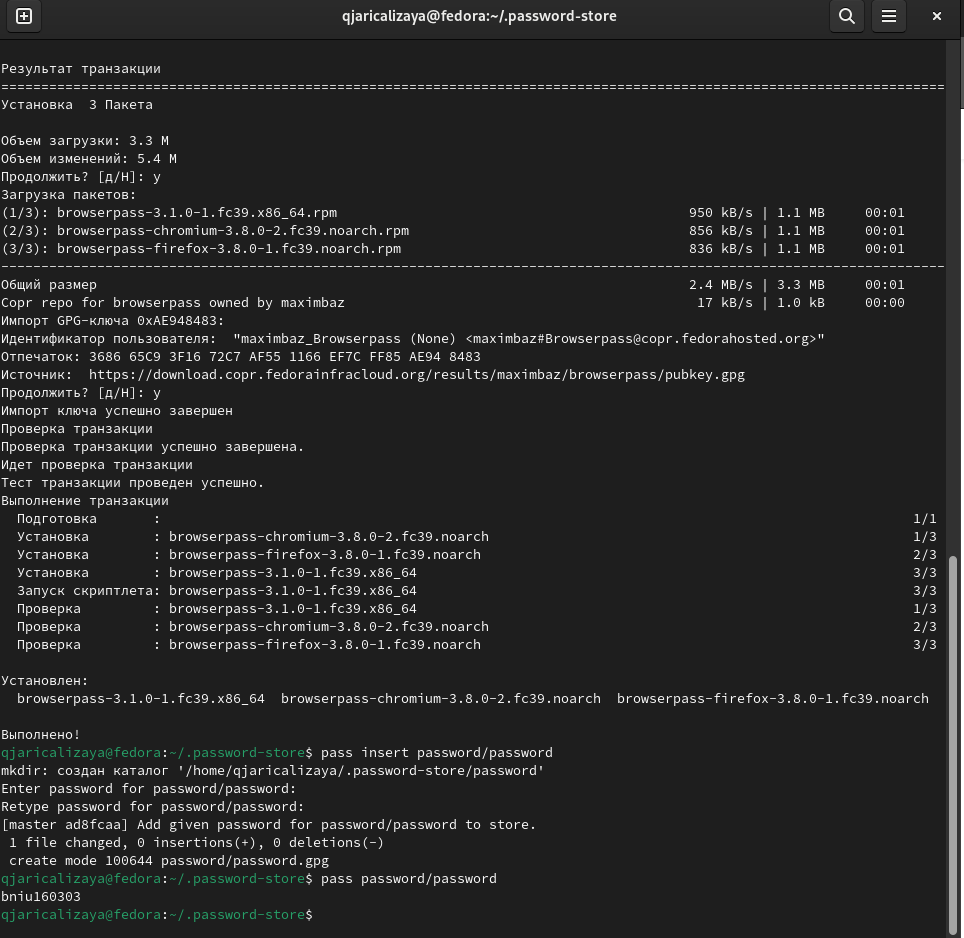


Рис. 12: пароль

И заменил существующий пароль (рис. 13).

pass generate --in-place password/password

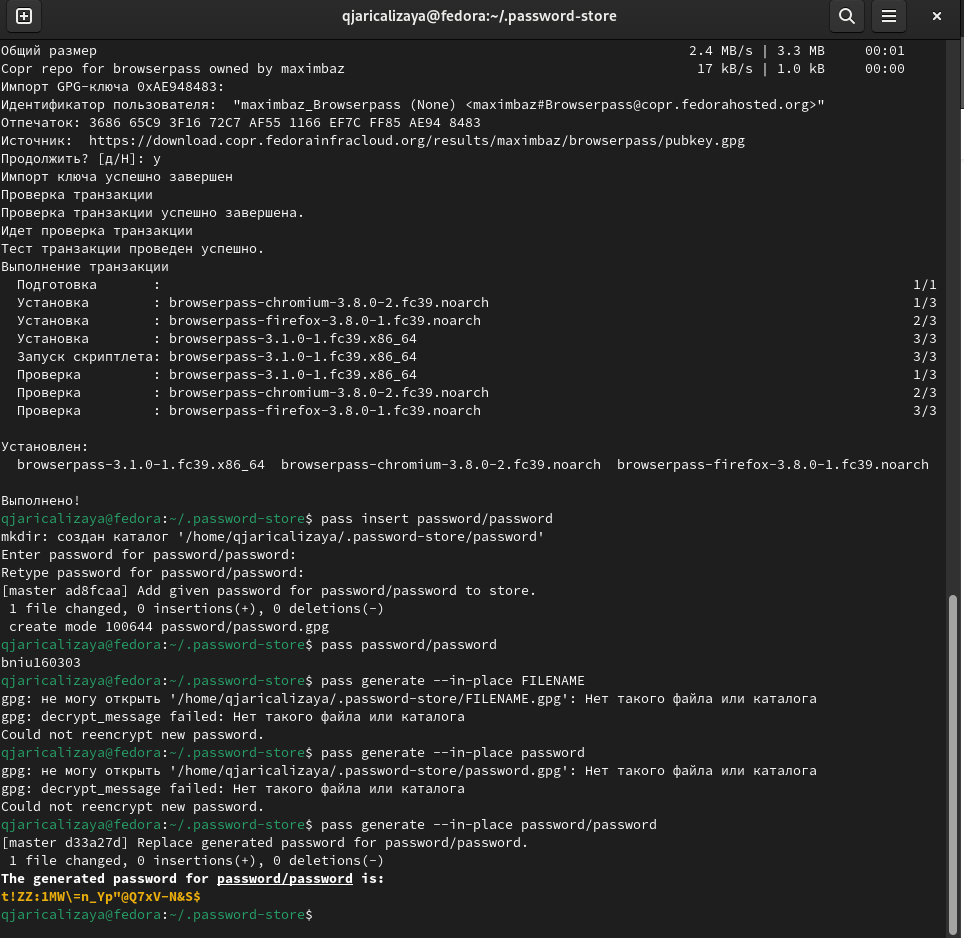


Рис. 13: изменение пароля

## 4.5 Управление файлами конфигурации

### 4.5.1 Дополнительное программное обеспечение

Я установил несколько программных обеспечений (рис. 14).

sudo dnf -y install \  
 dunst \  
 fontawesome-fonts \  
 powerline-fonts \  
 light \  
 fuzzel \  
 swaylock \  
 kitty \  
 waybar swaybg \  
 wl-clipboard \  
 mpv \  
 grim \  
 slurp

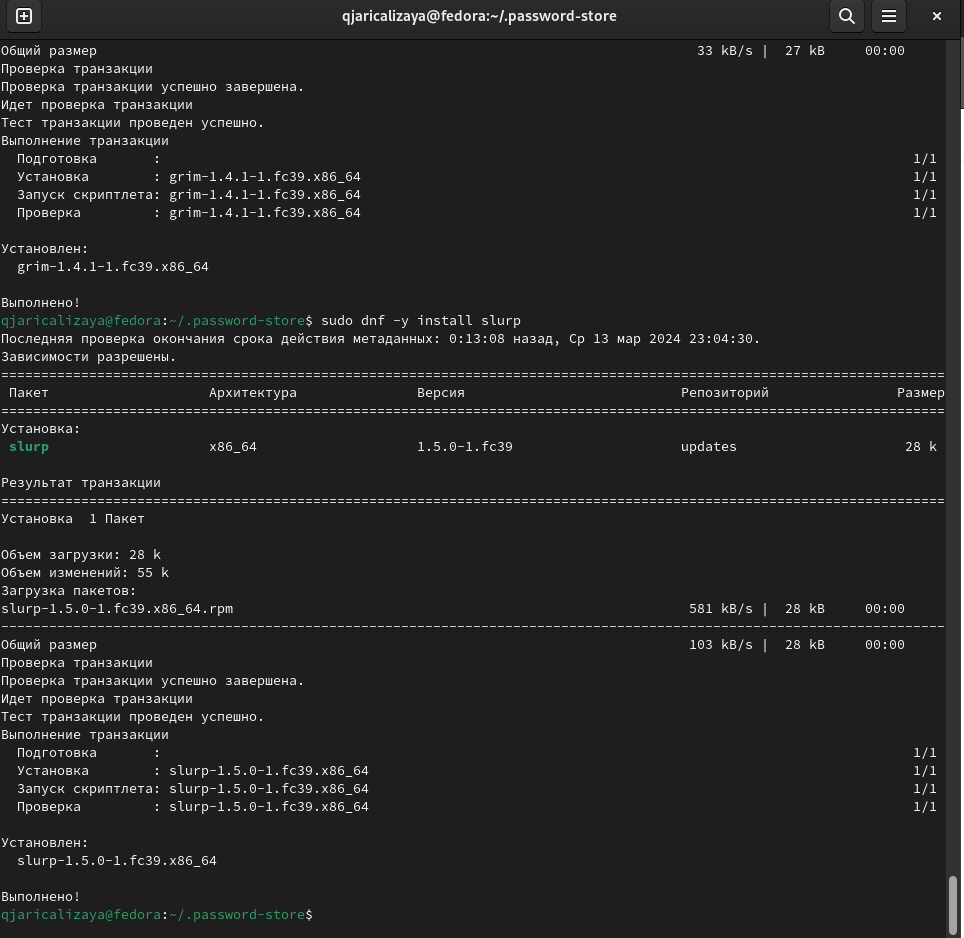


Рис. 14: установка программ

Потом я установил шрифты (рис. 15).

sudo dnf copr enable peterwu/iosevka  
 sudo dnf search iosevka  
 sudo dnf install iosevka-fonts iosevka-aile-fonts iosevka-curly-fonts iosevka-slab-fonts iosevka-etoile-fonts iosevka-term-fonts

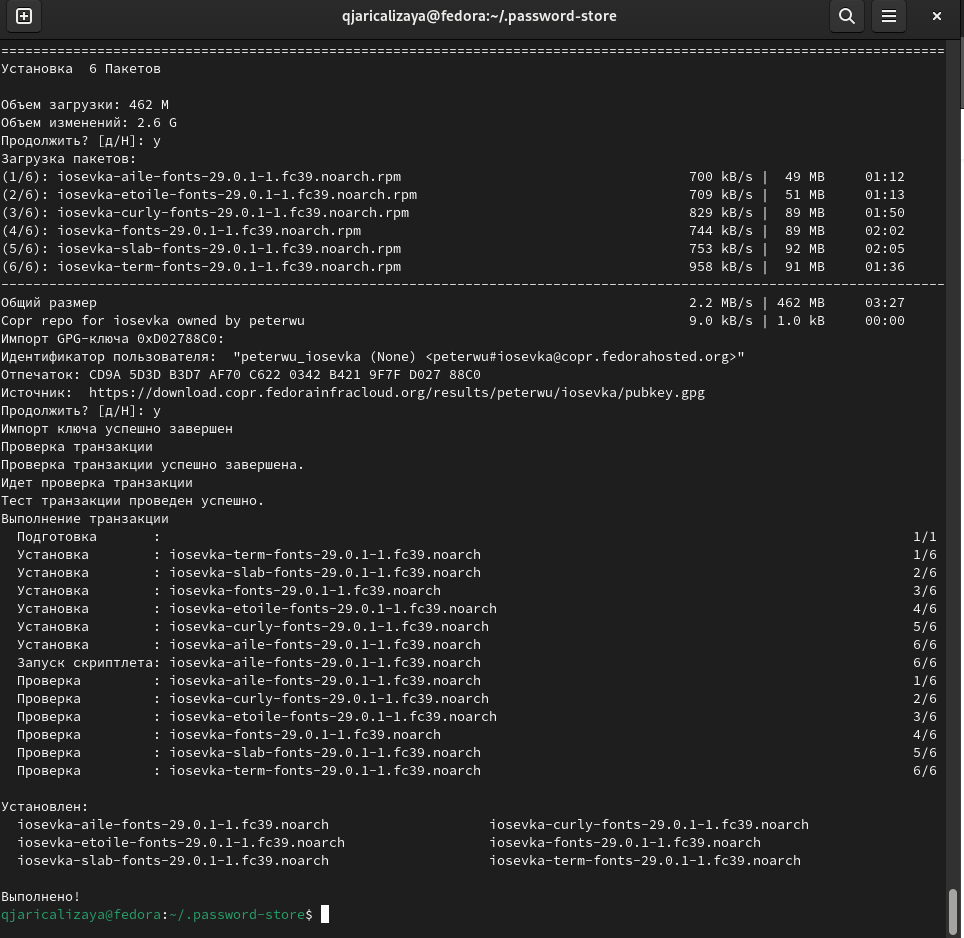


Рис. 15: установка программ

### 4.5.2 Установка

Я установил бинарный файл chezmoi (рис. 16).

sh -c "$(wget -qO- chezmoi.io/get)"

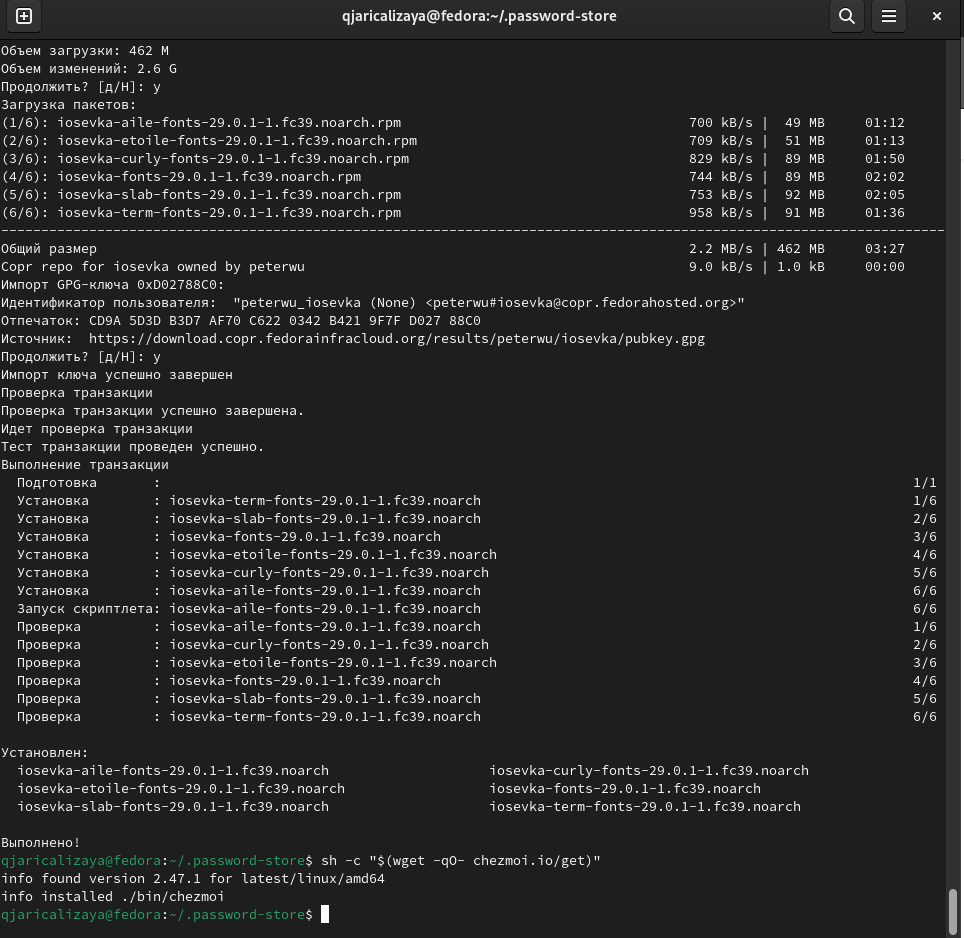


Рис. 16: установка chezmoi

### 4.5.3 Создание собственного репозитория с помощью утилит

Потом я копировал репозиторий dotfiles (рис. 17).

gh repo create dotfiles --template="yamadharma/dotfiles-template" --private

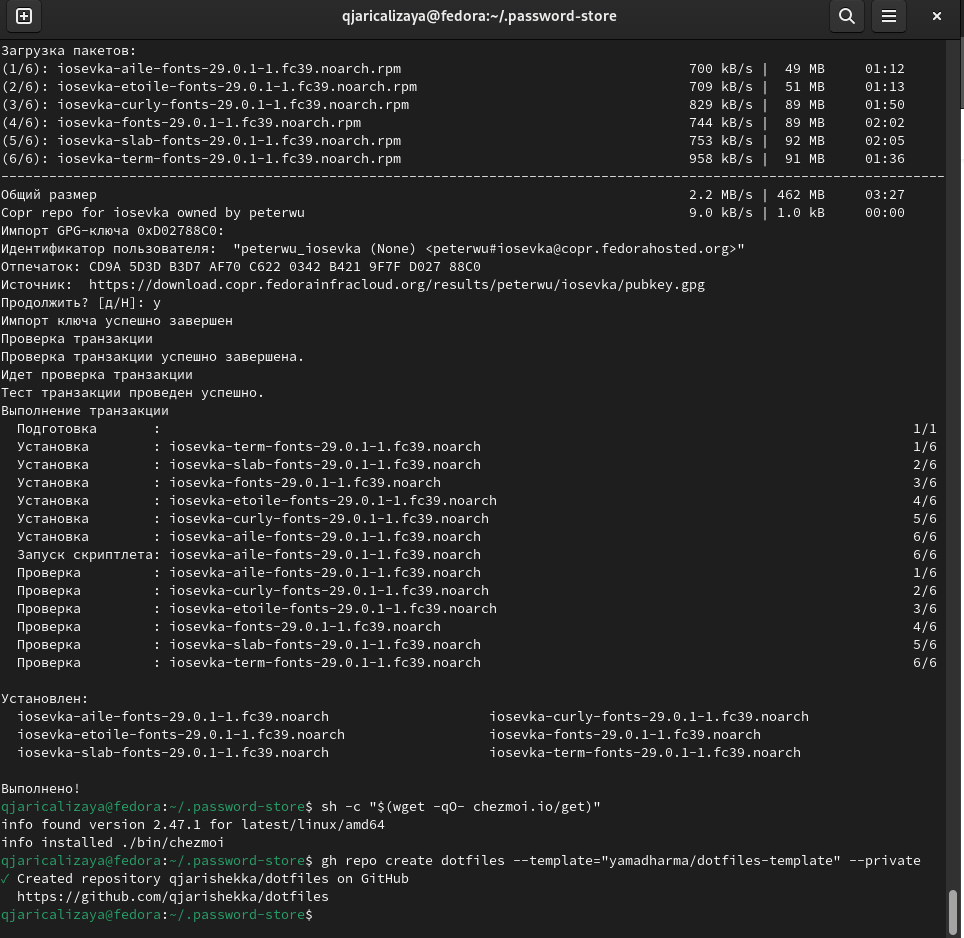


Рис. 17: копирование репозитория

### 4.5.4 Подключение репозитория к своей системе

Затем я инициализировал chezmoi (рис. 18).

chezmoi init git@github.com:qjarishekka/dotfiles.git

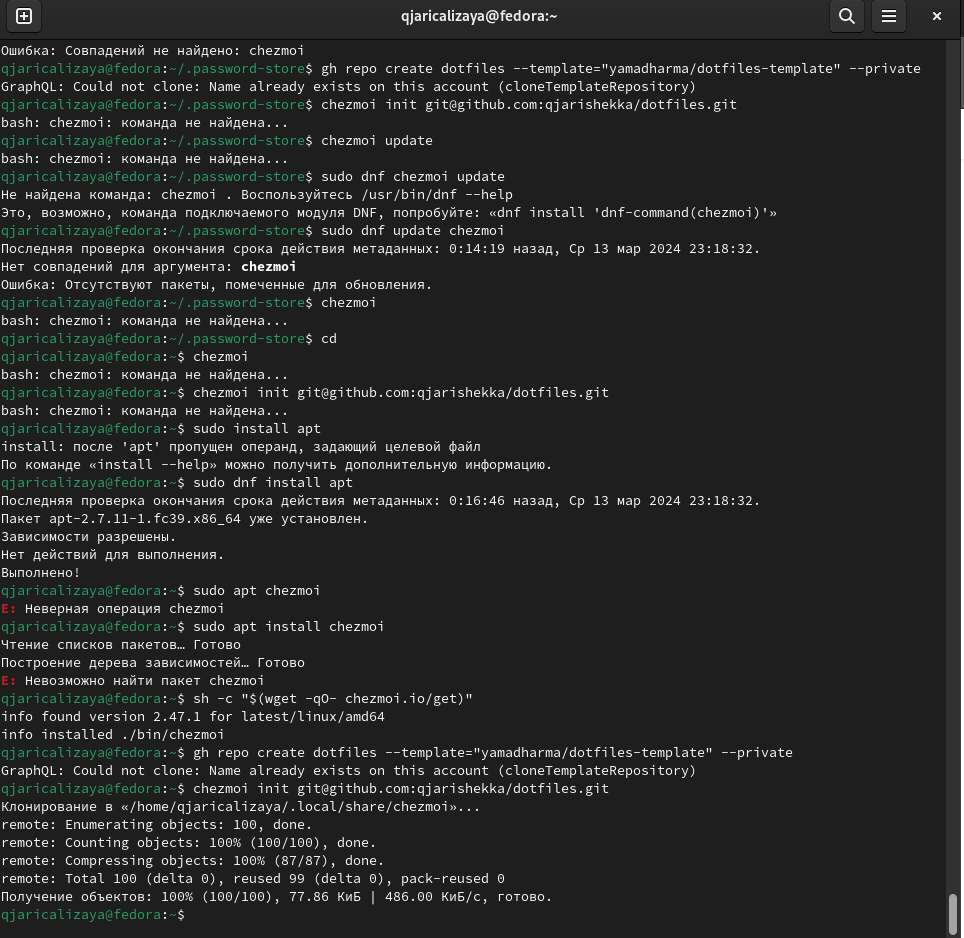


Рис. 18: инициализация репозитории

И проверал изменения (рис. 19).

chezmoi diff

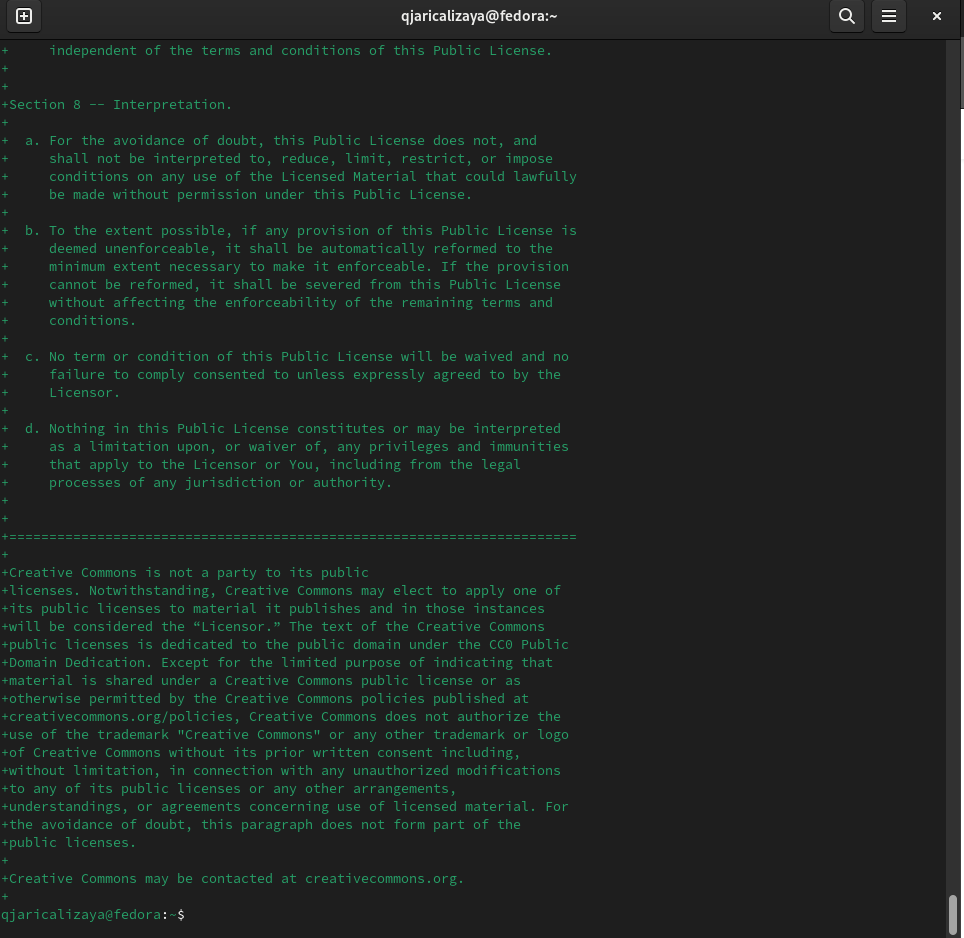


Рис. 19: проверка изменений

### 4.5.5 Использование chezmoi на нескольких машинах

Для выполнения этой части я включил новую виртуальную машину. установил chezmoi и несколько пакетов для правильно работы и инициализировал chezmoi там. (рис. 20).

chezmoi init git@github.com:qjarishekka/dotfiles.git

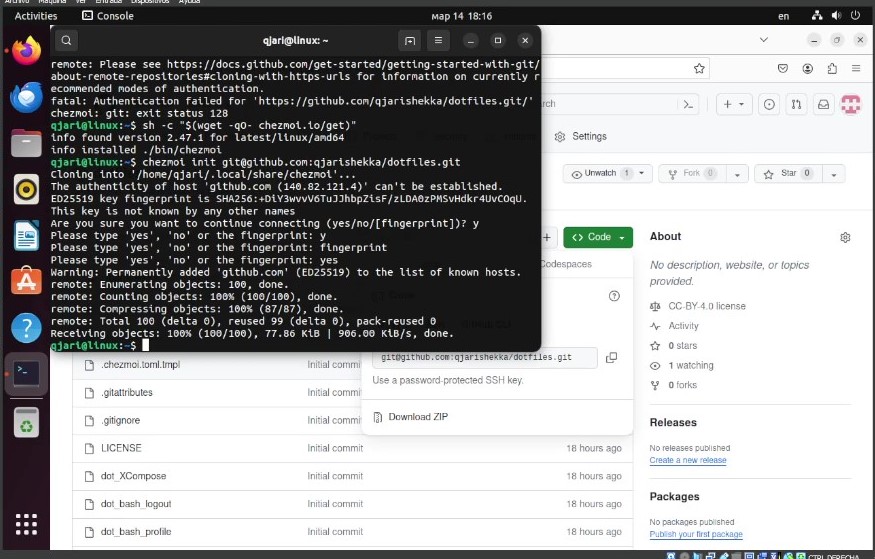


Рис. 20: инициализация репозитория

Затем я проверил изменения (рис. 21).

chezmoi diff

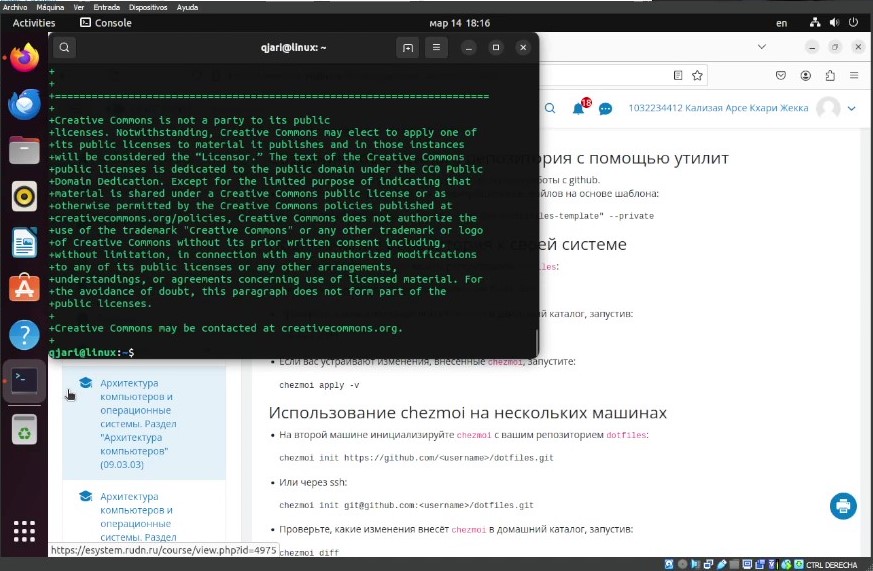


Рис. 21: проверка изменений

И запускал команду (рис. 22).

chezmoi apply -v

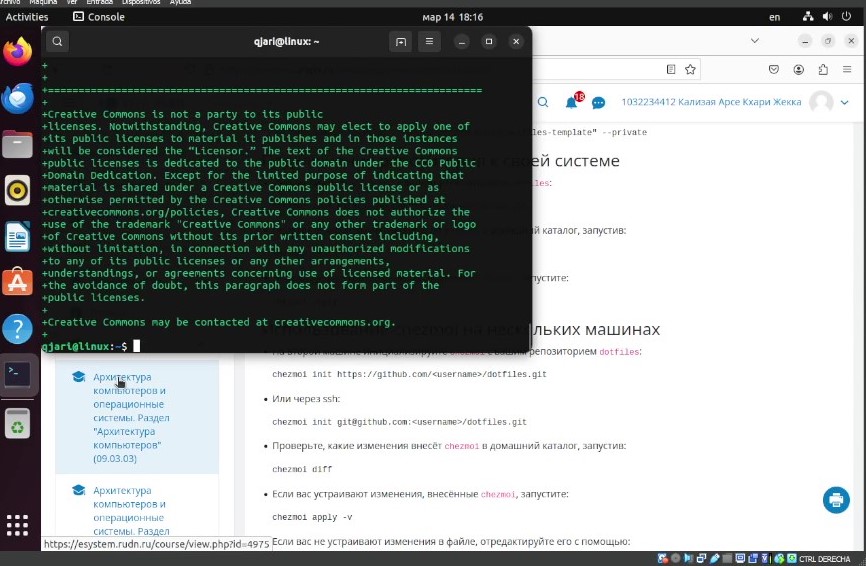


Рис. 22: принимать изменения

Потом я получил и применил последние изменения из моего репозитория (рис. 23).

chezmoi update -v

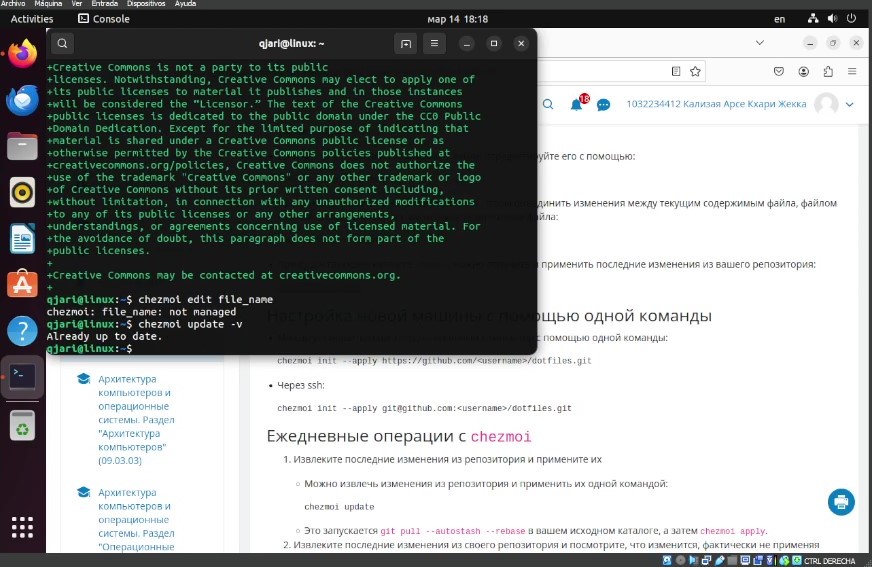


Рис. 23: принимать последние изменения

### 4.5.6 Ежедневные операции c chezmoi

Здесь я извлек изменения из репозитория и применил их (рис. 24).

chezmoi update

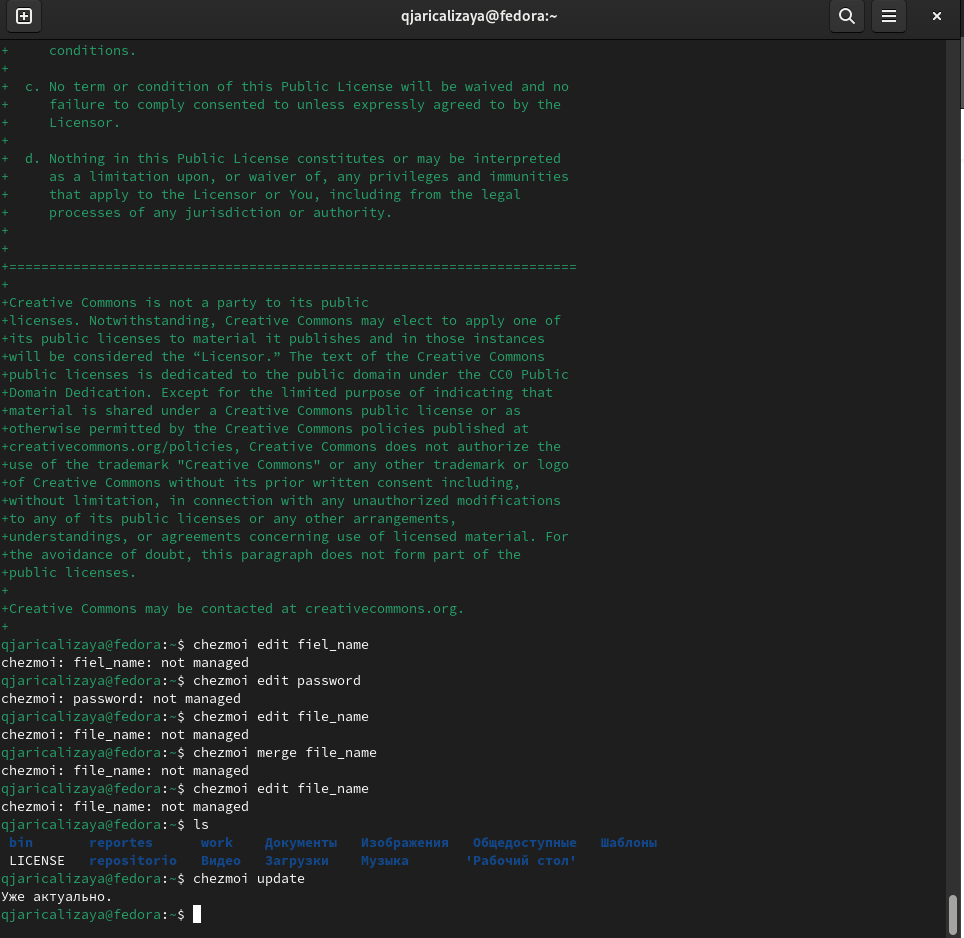


Рис. 24: Изменения

Потом я извлек последние изменения из своего репозитория. ничего изменился. .

chezmoi git pull -- --autostash --rebase && chezmoi diff

Затем я применил изменения (рис. 25)

chezmoi apply

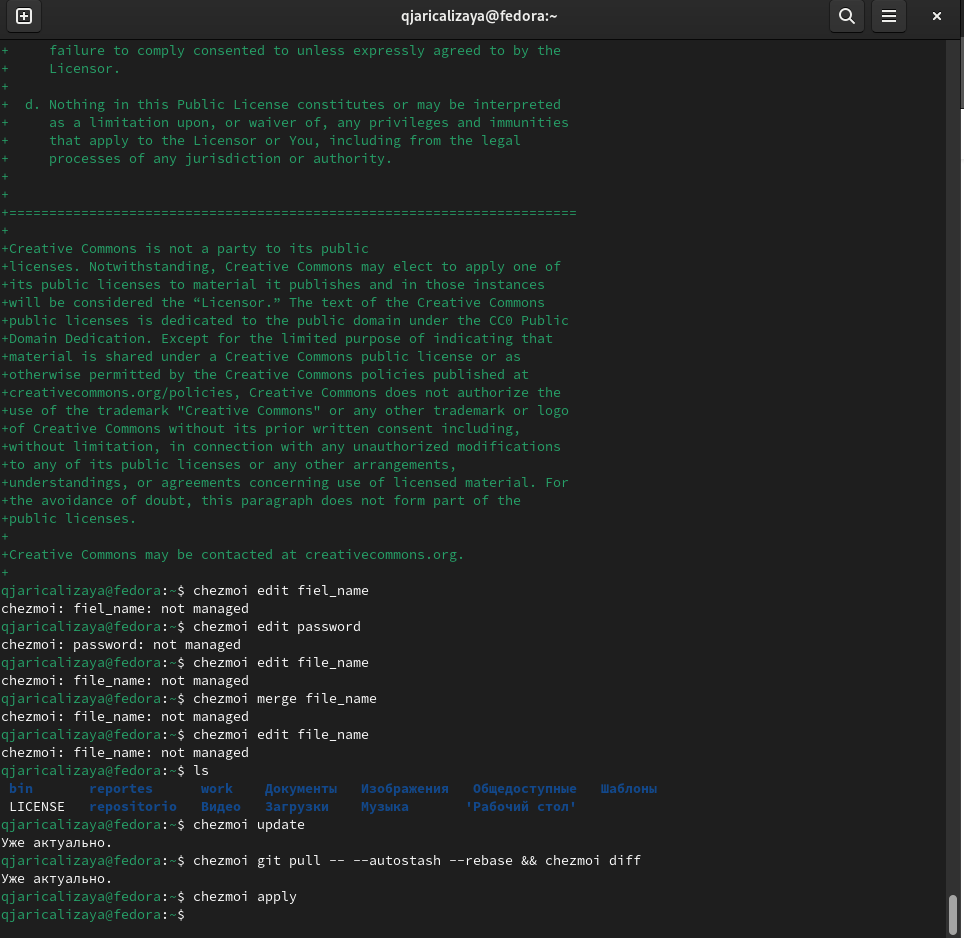


Рис. 25: Запуск на сервер

Дальше я проверял конфигурацию в файле ~/.config/chezmoi/chezmoi.toml (рис. 26).

sudo nano ~/.config/chezmoi/chezmoi.toml

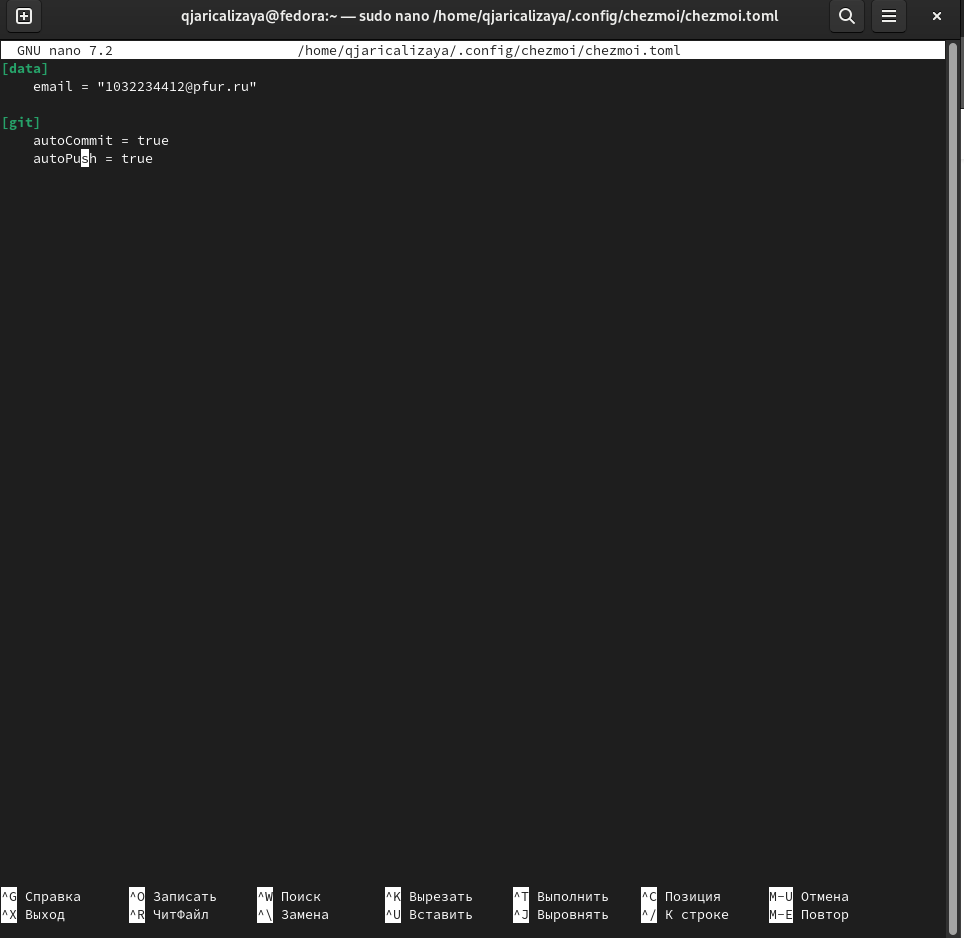


Рис. 26: проверка конфигурации

# 5 Выводы

В этой лабораторной работе, я смотрел процесс настройки и синхронизации репозитория с помощью chezmoi.

# Список литературы

1. Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. СПб.: Питер, 2015. 1120 с.

2. Robbins A. Bash Pocket Reference. O’Reilly Media, 2016. 156 с.

3. Zarrelli G. Mastering Bash. Packt Publishing, 2017. 502 с.

4. Newham C. [Learning the bash Shell: Unix Shell Programming](http://www.amazon.com/Learning-bash-Shell-Programming-Nutshell/dp/0596009658). O’Reilly Media, 2005. 354 с.