Отчёт по лабораторной работе №2

Дисциплина: Операционные системы

Шошина Евгения Александровна, группа: НКАбд-03-22

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

### 3.0.1 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.

# 4 Выполнение лабораторной работы

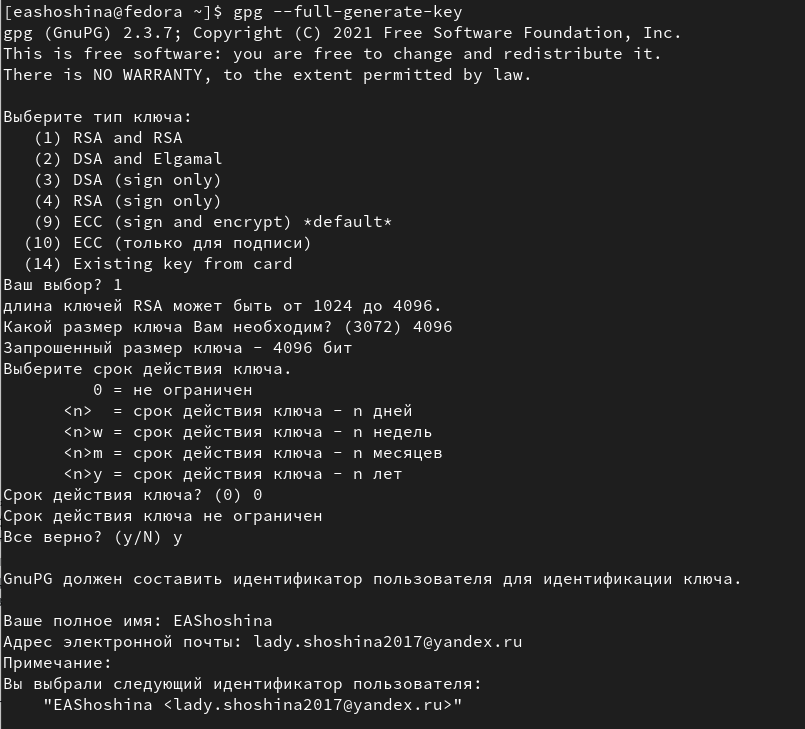
### 4.0.1 Установка программного обеспечения

Установка git - dnf install git Установка gh - dnf install gh Рис. 1.2 Установка gh - dnf install gh ### Базовая настройка git Зададим имя и email владельца репозитория: Рис. 2.2 Имя и email владельца репозитория Настроим utf-8 в выводе сообщений git: Рис. 3.2 utf-8 в выводе сообщений git Настройте верификацию и подписание коммитов git (см. Верификация коммитов git с помощью GPG). Зададим имя начальной ветки (будем называть её master): Рис. 4.2 имя начальной ветки Параметр autocrlf: Рис. 5.2 Параметр autocrlf Параметр safecrlf: Рис. 6.2 Параметр safecrlf:

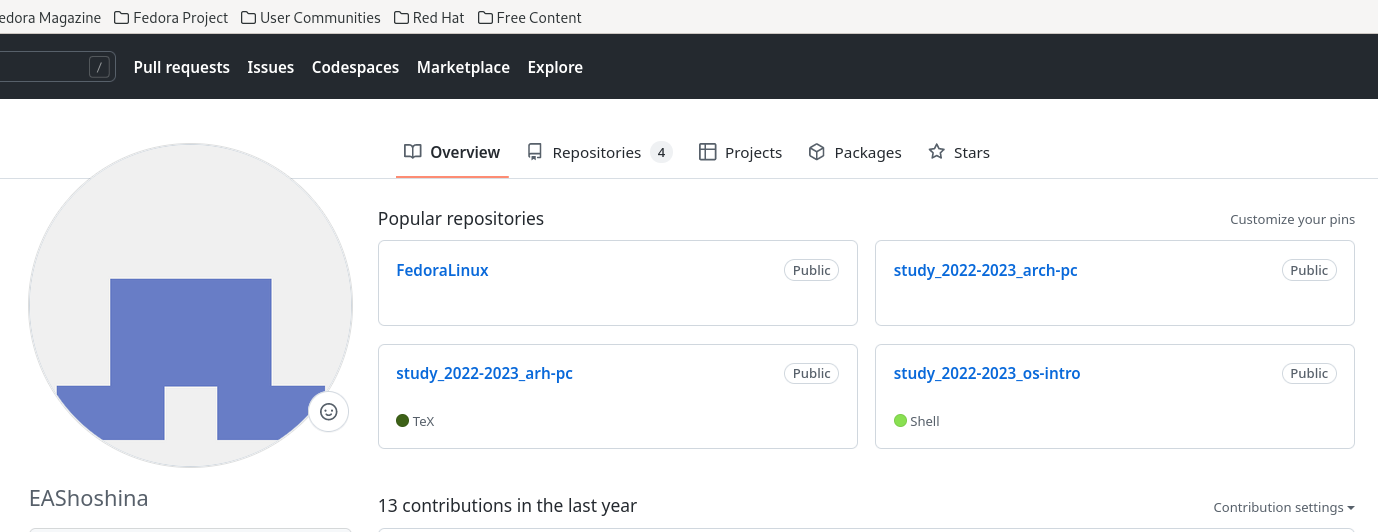
### 4.0.2 Создали ключи ssh

по алгоритму rsa с ключом размером 4096 бит(ssh-keygen -t rsa -b 4096) по алгоритму ed25519(ssh-keygen -t ed25519)

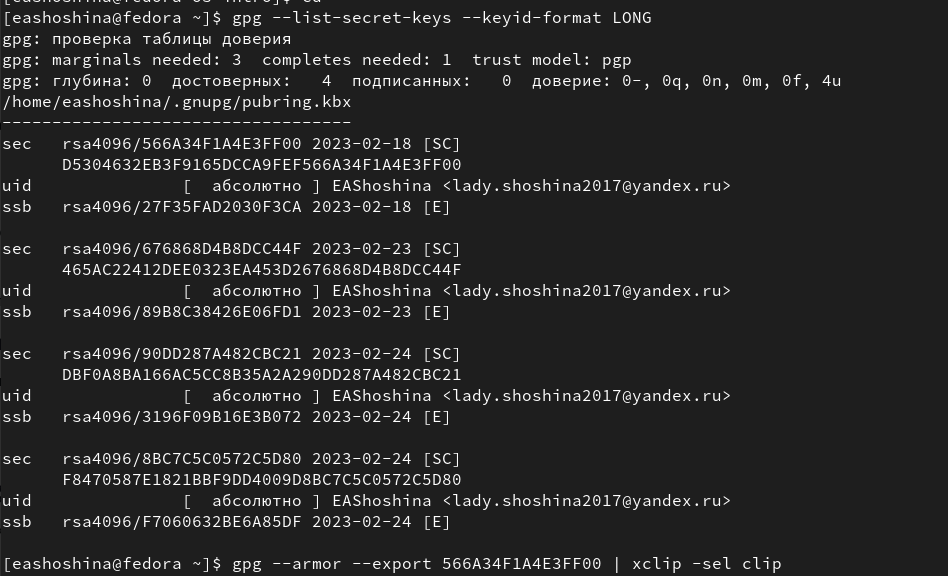
### 4.0.3 Создали ключи pgp

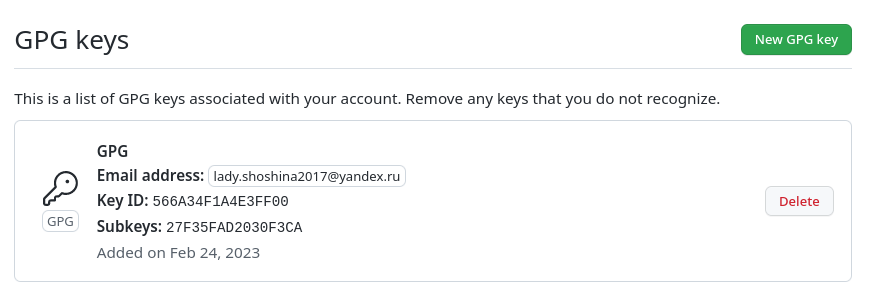
Генерируем ключ 

### 4.0.4 Настройка github

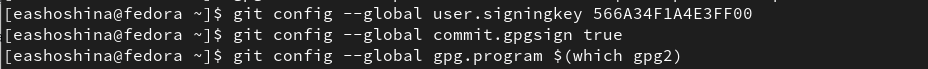
Создали учётную запись на https://github.com. Заполнили основные данные на https://github.com. 

### 4.0.5 Добавили PGP ключ в GitHub

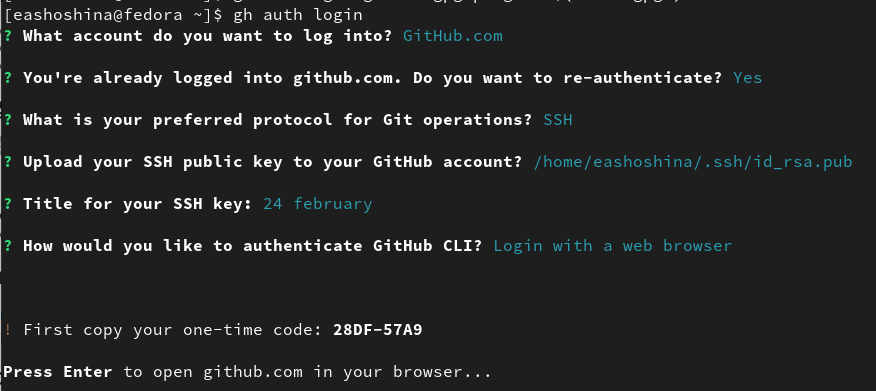
Выводим список ключей и копируем отпечаток приватного ключа: Cкопировали наш сгенерированный PGP ключ в буфер обмена: 

Перешли в настройки GitHub (https://github.com/settings/keys), нажали на кнопку New GPG key и вставили полученный ключ в поле ввода. 

### 4.0.6 Настройка автоматических подписей коммитов git

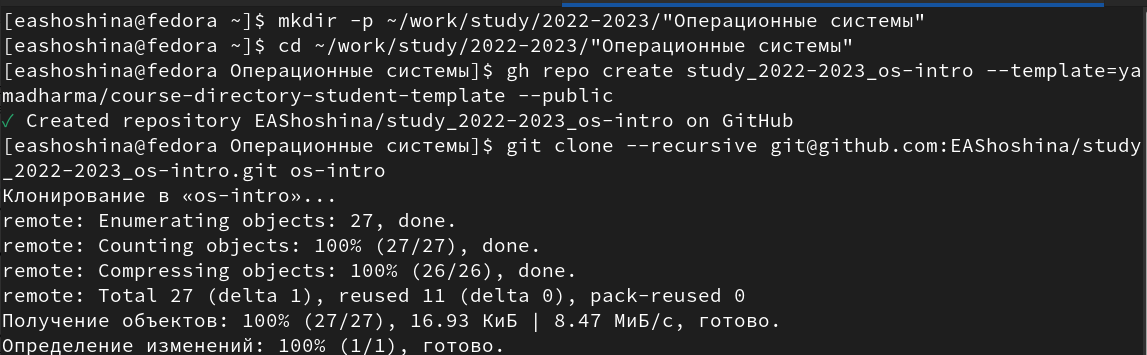
Используя введёный email, указали Git, чтобы применять при подписи коммитов: 

### 4.0.7 Настройка gh

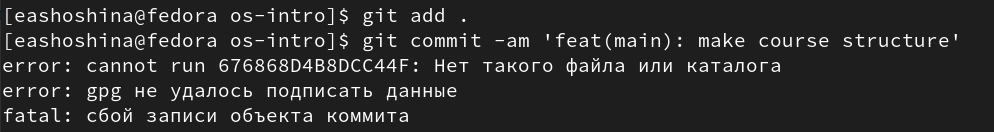
Для начала авторизовались 

### 4.0.8 Сознание репозитория курса на основе шаблона

Необходимо создать шаблон рабочего пространства (см. Рабочее пространство для лабораторной работы).

Например, для 2022–2023 учебного года и предмета «Операционные системы» (код предмета os-intro) создание репозитория примет следующий вид: 

### 4.0.9 Настройка каталога курса

Перешли в каталог курса: Удалили лишние файлы: Создайте необходимые каталоги:  Отправьте файлы на сервер: 

# 5 Выводы

1. Изучили идеологию и применение средств контроля версий.
2. Освоили умения по работе с git.

# 6 Контрольные вопросы

#### 6.0.0.1 1) Система контроля версий (Version Control System, VCS)

— программное обеспечение для облегчения работы с изменяющейся информацией. #### Система контроля версий (Version Control System, VCS) — это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов. VCS 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) Хранилище (repository), или репозиторий, — место хранения файлов и их версий, служебной информации. #### Commit («[трудовой] вклад», не переводится) — процесс создания новой версии; иногда синоним версии. #### Версия (revision), или ревизия, — состояние всего хранилища или отдельных файлов в момент времени («пункт истории»). #### Рабочая копия (working copy) — текущее состояние файлов проекта (любой версии), полученных из хранилища и, возможно, измененных.

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)Централизованные VCS: Одно основное хранилище всего проекта. Каждый пользователь копирует себе необходимые ему файлы из этого репозитория, изменяет и, затем, добавляет свои изменения обратно (Subversion, CVS, TFS, VAULT, AccuRev) #### Децентрализованные VCS: У каждого пользователя свой вариант (возможно не один) репозитория Присутствует возможность добавлять и забирать изменения из любого репозитория (Git, Mercurial, Bazaar) #### 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: git push origin имя\_ветки – слияние ветки с текущим деревом: 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. # Список литературы{.unnumbered}