Отчёт по лабораторной работе №2

Управление версиями

Великоднева Евгения

Содержание

# Цель работы

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

# 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.

# Теоретическое введение

Системы контроля версий (Version Control System, VCS) применяются при работе несколь- ких 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.

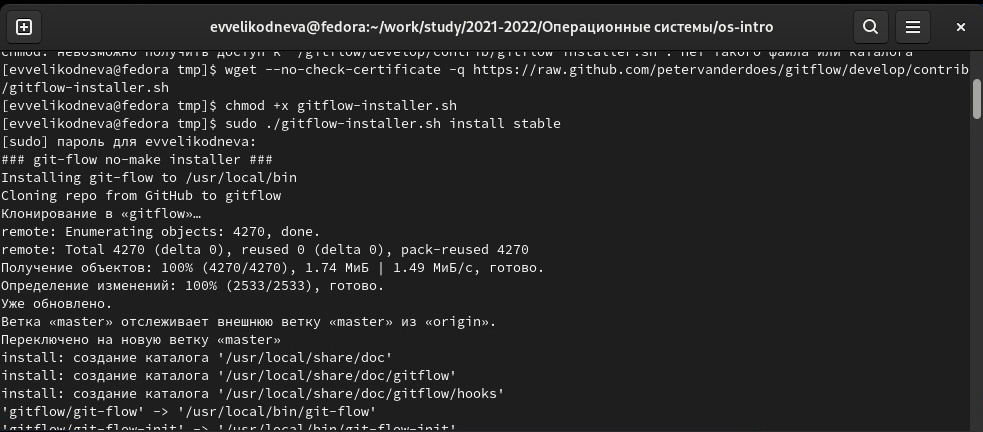
Система контроля версий Git представляет собой набор программ командной строки. Доступ к ним можно получить из терминала посредством ввода команды git с различ- ными опциями.

Для последующей идентификации пользователя на сервере репозиториев необходимо сгенерировать пару ключей (приватный и открытый). Ключи сохраняться в каталоге ~/.ssh/.

– Gitflow Workflow опубликована и популяризована Винсентом Дриссеном из компании vie. – Gitflow Workflow предполагает выстраивание строгой модели ветвления с учётом выпуска проекта. – Данная модель отлично подходит для организации рабочего процесса на основе рели- зов. – Работа по модели Gitflow включает создание отдельной ветки для исправлений ошибок в рабочей среде.

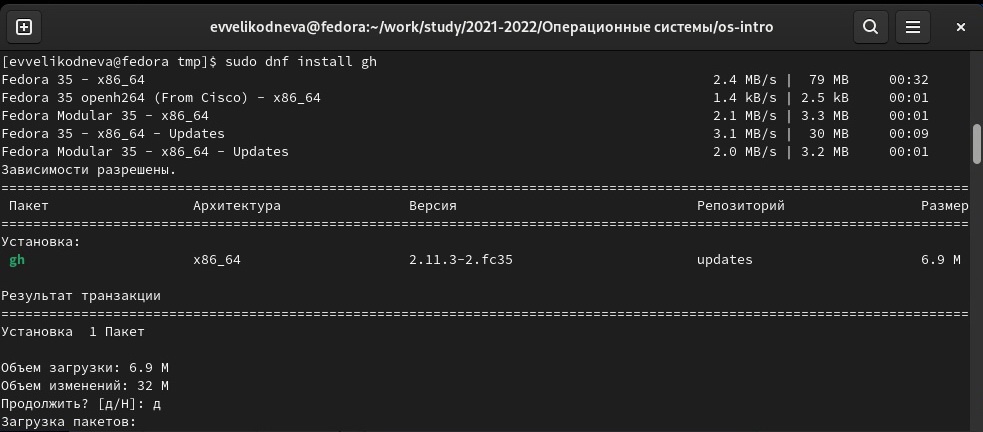
# Выполнение лабораторной работы

1. Создала учётную запись на https://github.com и заполнила основные данные.
2. Перешла в каталог /tmp с помощью команды cd и установила необходимое программное обеспечение. (рис. [-@fig:001])



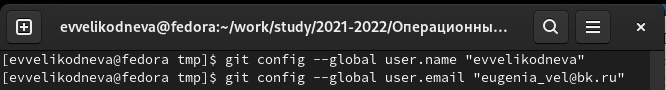
Установка необходимого программного обеспечения

1. Установила gh с помощью команды dnf install. (рис. [-@fig:002])



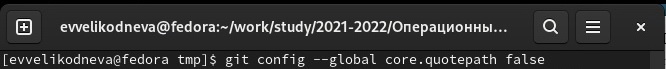
Установка gh

1. Задала имя и email владельца репозитория. (рис. [-@fig:003])



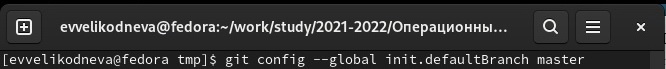
Ввод имени и электронной почты владельца репозитория

1. Настроила utf-8 в выводе сообщений git. (рис. [-@fig:004])



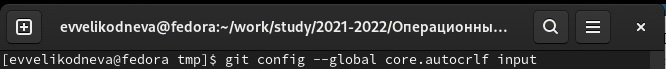
Настройка utf-8 в выводе сообщений git

1. Настроила верификацию и подписание коммитов git, задала имя начальной ветки master. (рис. [-@fig:005])



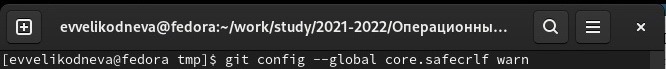
Настройка верификации и подписания коммитов, задание имени начальной ветки

1. Настроила параметр autocrlf. (рис. [-@fig:006])



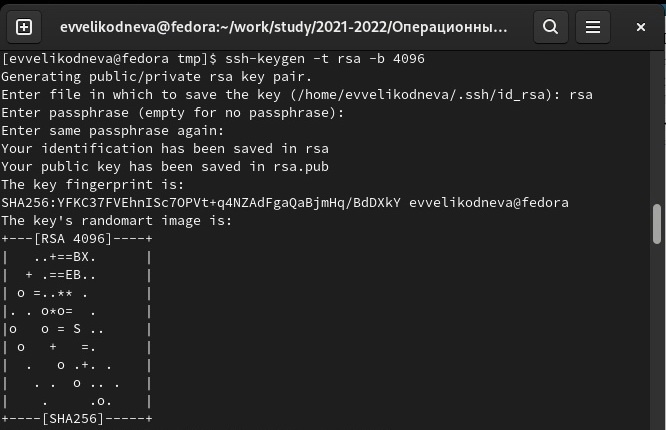
Настройка параметра autocrlf

1. Настроила параметр safecrlf. (рис. [-@fig:007])



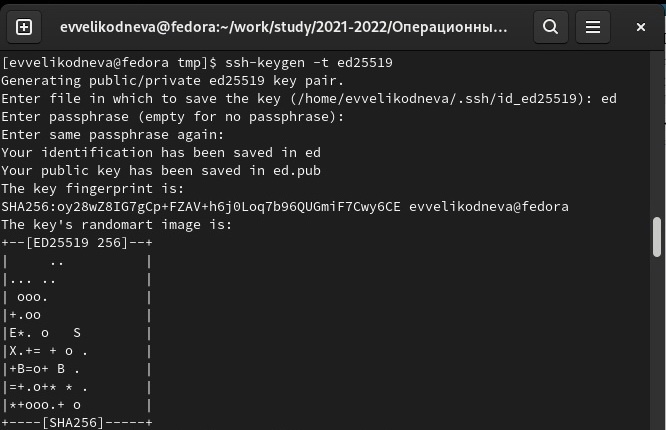
Настройка параметра safecrlf

1. Создала ssh ключ по алгоритму rsa. (рис. [-@fig:008])



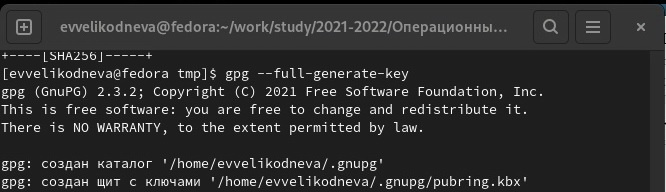
Создание ssh ключа по алгоритму rsa

1. Создала ssh ключ по алгоритму ed25519. (рис. [-@fig:009])



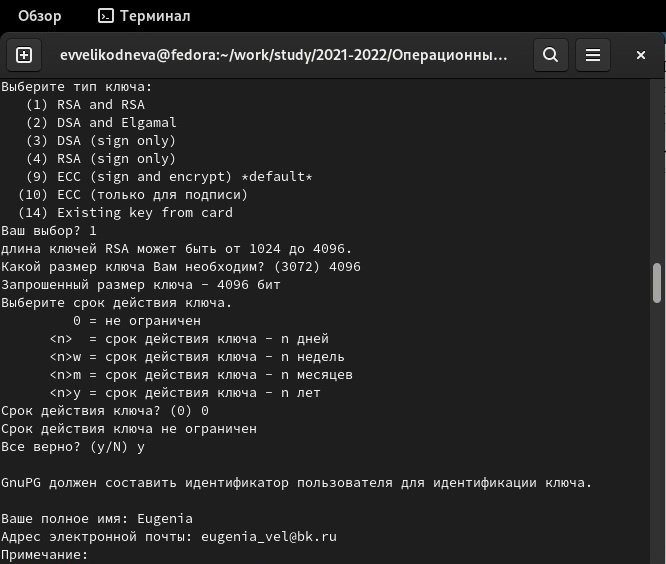
Создание ssh ключа по алгоритму ed25519

1. Сгенерировала ключ. (рис. [-@fig:010])

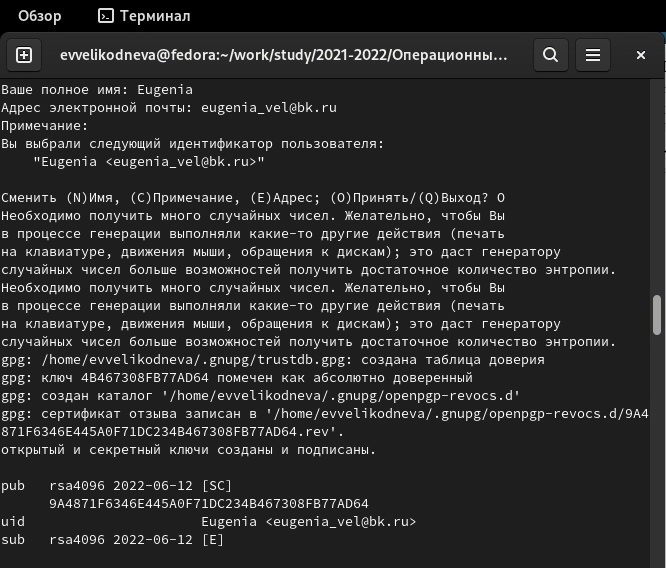


Генерация ключа

1. Выбрала тип ключа RSA and RSA, задала размер 4096, выбрала срок действия по умолчанию. Ввела запрашиваемую информацию: имя, адрес электронной почты, примечание оставила пустым. (рис. [-@fig:011]) Мне предложили поменять данные, которые я ввела, но так как всё было правильно, я ничего больше не меняла. (рис. [-@fig:012])

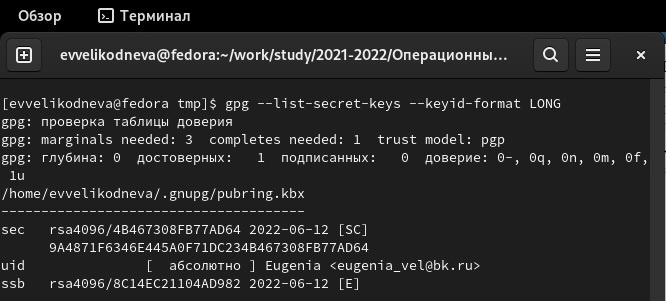


Выбор типа ключа, размера, срока действия. Ввод запрашиваемой информации



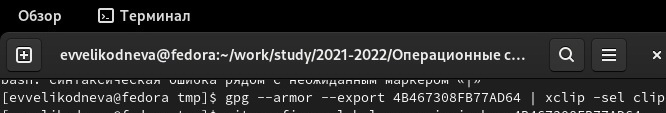
Создание ключа

1. Вывела список ключей и скопировала отпечаток приватного ключа. (рис. [-@fig:013])



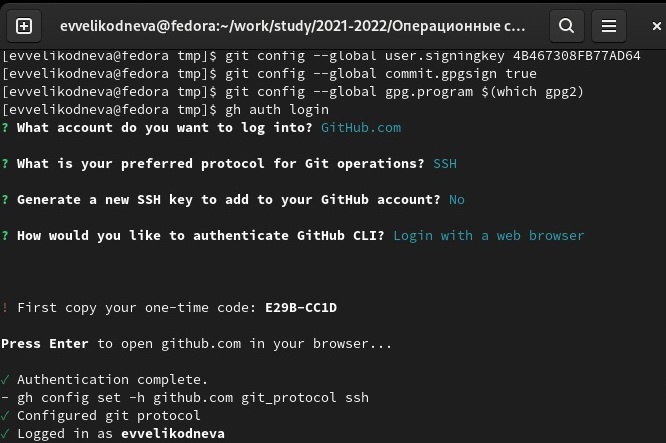
Вывод списка ключей

1. Скопировала сгенерированный PGP ключ в буфер обмена, перешла в настройки GitHub и добавила новый PGP ключ. (рис. [-@fig:014])



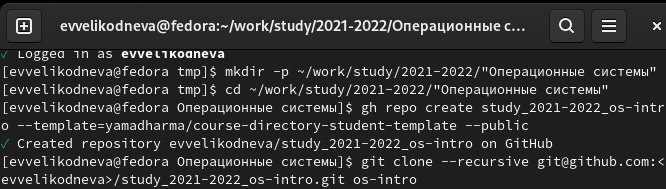
Копирование сгенерированного PGP ключа

1. Используя введёный email, указала Git применять его при подписи коммитов. (рис. [-@fig:015])



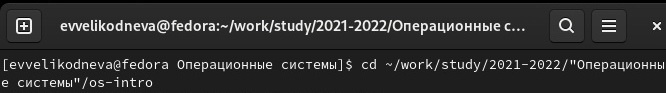
Указала Git применять введённый email при подписи коммитов, авторизовалась в gh

1. Авторизовалась в gh. (рис. [-@fig:015])
2. Клонировала шаблон репозитория к себе в gh, а затем скопировала его к себе в систему. (рис. [-@fig:016])



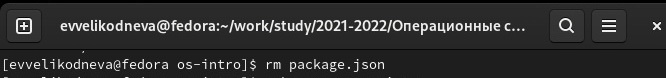
Клонирование шаблона репозитория и копирование его к себе в ПК

1. Перешла в каталог курса с помощью команды cd. (рис. [-@fig:017])



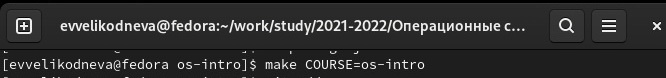
Переход в каталог курса

1. Удалила лишние файлы с помощью команды rm. (рис. [-@fig:018])



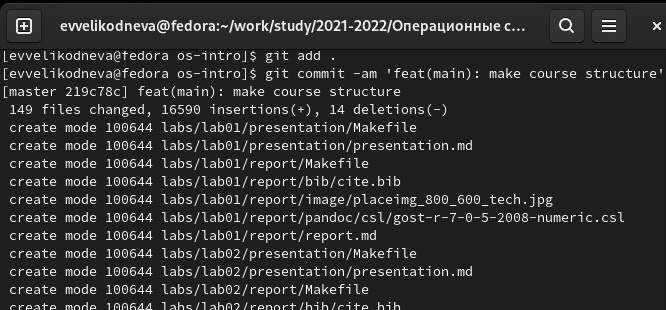
Удаление лишних файлов

1. Создала необходимые каталоги с помощью команды make. (рис. [-@fig:019])



Создание необходимых каталогов

1. Отправила файлы на сервер с помощью команд git add, git commit и git push. (рис. [-@fig:020])



Отправка файлов на сервер

# Контрольные вопросы

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

Системы контроля версий – это набор программного обеспечения, они предназначаются для работы нескольких человек над одним проектом

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

• Хранилище — сервер, на котором хранится вся история изменений проекта. • Commit — фиксация “дельта-изменений”, т.е. изменений с последнего commit’a с его последующей записью как версии в истории. • История — список всех изменений проекта с возможностью отката в любую точку истории. • Рабочая копия — все файлы проекта, с которыми происходит основная работа.

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

В централизованных VCS необходим центральный репозиторий для хранения файлов. Примером таковых могут служить CVS и Subversion. В децентрализованных VCS наличие центрального репозитория не обязательно. Децентрализованными VCS являются Git, Bazaar и Mercurial.

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

Инициализация системы управления версиями git через git init. Работа над проектом используя git-flow для отдельных частей проекта. Git commit для фиксации изменений. При необходимости использование удаленного сервера для хранения с помощью remote и git push. Удаленный сервер также позволяет работать с нескольких устройств с использованием git pull.

1. Опишите порядок работы с общим хранилищем VCS.

При существующей версии проекта в хранилище, скопировать его оттуда через git pull. Использовать git-flow для работы над частями проекта. После окончания работы зафиксировать изменения через git commit и загрузить в хранилище через git push.

1. Каковы основные задачи, решаемые инструментальным средством git? Ведение истории изменений, фиксирование изменений, совмещение версий, веток и тд. Откат к прошлым версиям.
2. Назовите и дайте краткую характеристику командам git. • git init — инициализация проекта с системой контроля версий • git add — добавление файла/директории в систему контроля версий как отслеживаемое • git commit — фиксация изменений в отслеживаемых файлах • git push — загрузка локальной версии на сервер • git pull — выгрузка актуальной версии с сервера • git fetch — “часть” команды git pull, которая собирает актуальную версию, но не вносит её в работу • git merge — слияние веток
3. Приведите примеры использования при работе с локальным и удалённым репозиториями.

Для того, чтобы просмотреть список настроенных удалённых репозиториев, вы можете запустить команду git remote. Она выведет названия доступных удалённых репозиториев. Если вы клонировали репозиторий, то увидите, как минимум origin — имя по умолчанию, которое Git даёт серверу, с которого производилось клонирование. Получение изменений из удалённого репозитория — Fetch и Pull Для того, чтобы добавить удалённый репозиторий и присвоить ему имя (shortname), просто выполните команду git remote add .

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

Ветка — 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?

Во время работы над проектом так или иначе могут создаваться файлы, которые не требуется добавлять в последствии в репозиторий. Например, временные файлы, создаваемые редакторами, или объектные файлы, создаваемые компиляторами. Можно прописать шаблоны игнорируемых при добавлении в репозиторий типов файлов в файл .gitignore с помощью сервисов.

# Выводы

Изучила идеологию и применение средств контроля версий, освоила умения по работе с git.