Отчет по лабораторной работе №2

Дисциплина: Операционные системы

Иванов Сергей Владимирович

Содержание

Список иллюстраций

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 Выполнение лабораторной работы

Установим git ‘dnf install git’(рис. 1).

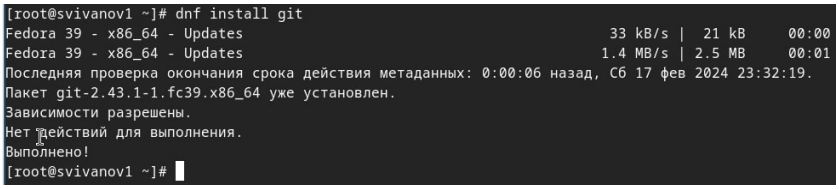

Рис. 3.1: Установка git

Установим gh ‘dnf install gh’ (рис. 2).

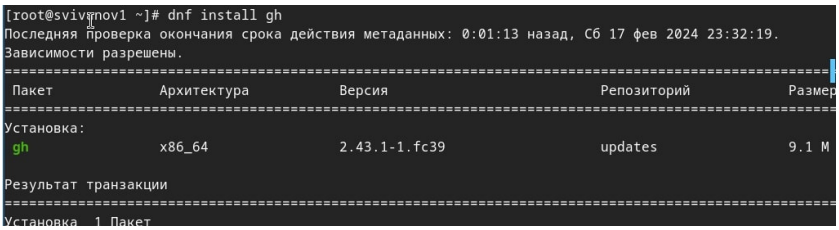

Рис. 3.2: Установка gh

Проведем базовую настройку git. Зададим имя и email владельца репозитория, настроим utf-8 в выводе сообщений git, зададим имя начальной ветки, укажем параметр autocrlf, параметр safecrlf (рис. 3).

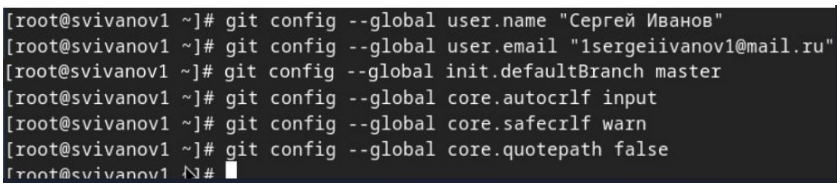

Рис. 3.3: Базовая настройка git

Создадим ключ ssh по алгоритму rsa с ключём размером 4096 бит ‘ssh-keygen -t rsa -b 4096’ (рис. 4).

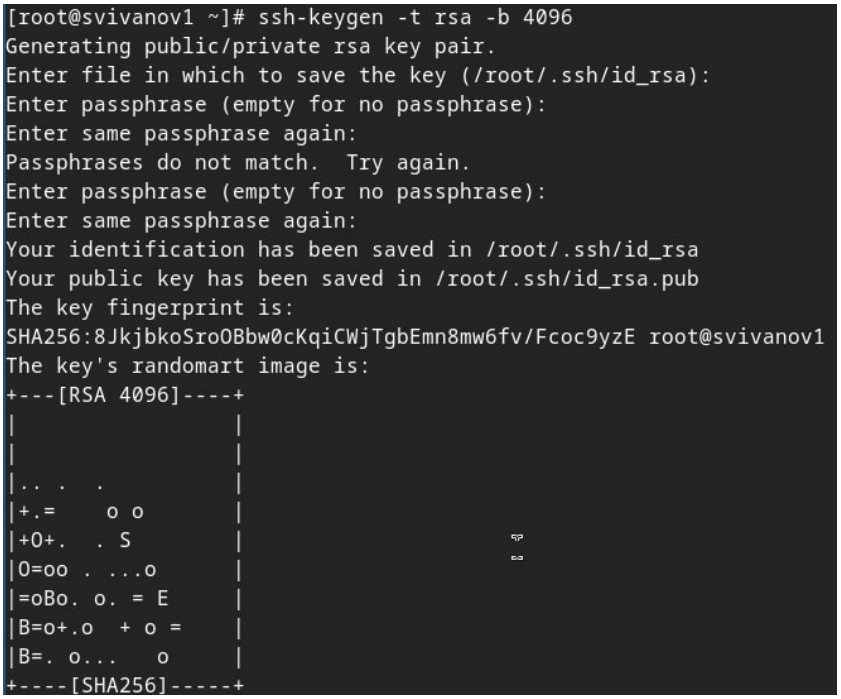

Рис. 3.4: Алгоритм rsa

Создадим ключ ssh по алгоритму ed25519 ‘ssh-keygen -t ed25519’ (рис. 5).

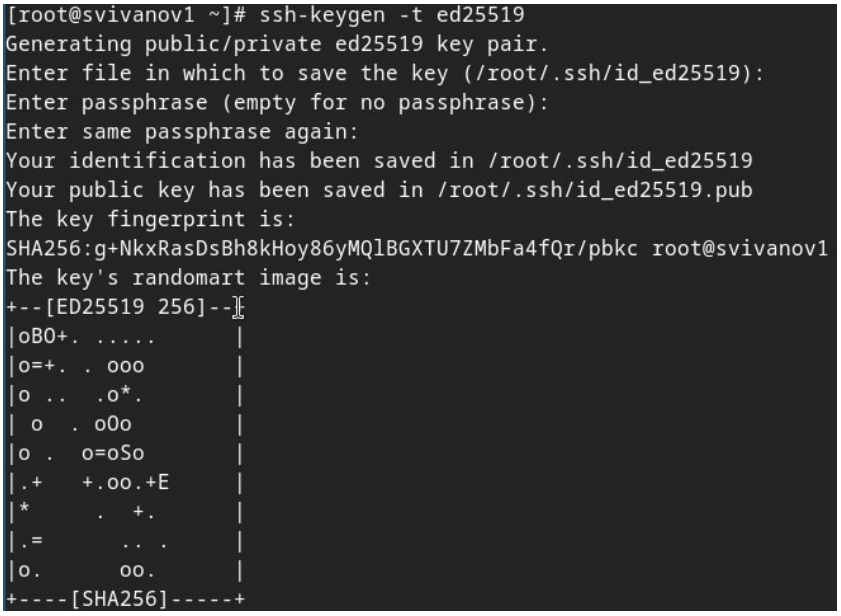

Рис. 3.5: Алгоритм ed25519

Создадим ключ pgp ‘gpg –full-generate-key’, выбираем тип RSA and RSA, размер 4096, срок действия - не истекает никогда. (рис. 6).

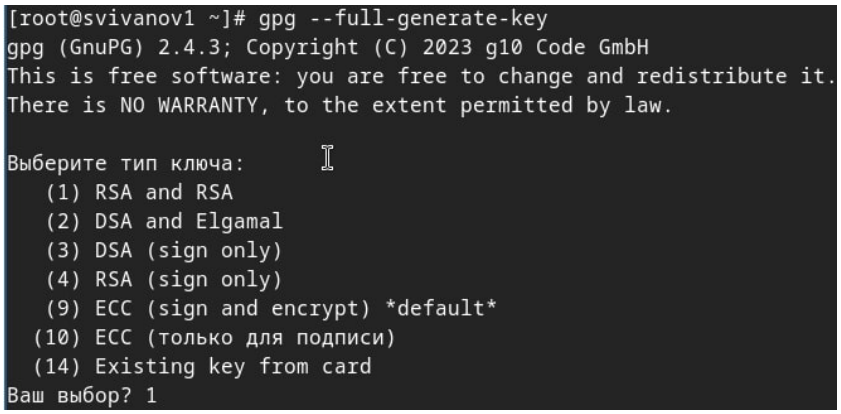

Рис. 3.6: Ключ pgp

Пропускаем этап создания учетной записи GitHub, так как она уже создана

Выводим список ключей и копируем отпечаток приватного ключа ‘gpg –list-secret-keys –keyid-format LONG’, скопируем наш сгенерированный PGP ключ в буфер обмена (рис. 7).

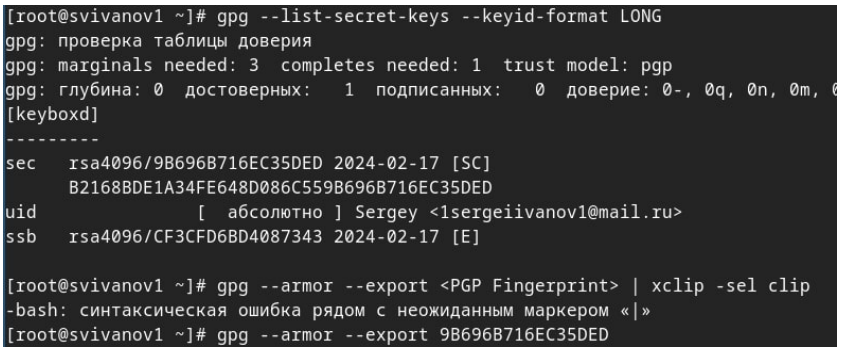

Рис. 3.7: Копируем ключ

Перейдём в настройки GitHub (https://github.com/settings/keys), нажмем на кнопку New GPG key и вставим полученный ключ в поле ввода. (рис. 8).

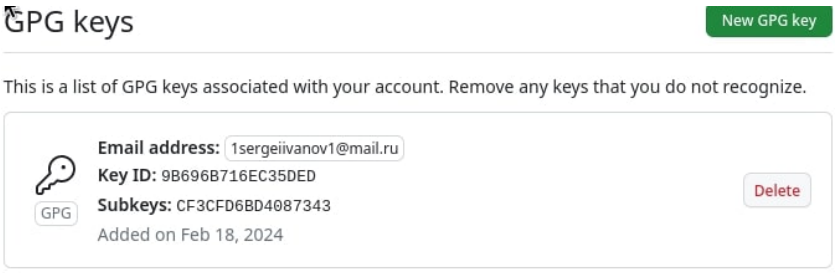

Рис. 3.8: Вставляем ключ

Настроим автоматические подписи коммитов git Используя введёный email, укажем Git применять его при подписи коммитов: (рис. 9).

```
[root@svivanov1 ~]# git config --global user.signingkey 9B696B716EC35DED
[root@svivanov1 ~]# git config --global commit.gpgsign tru
[root@svivanov1 ~]# git config --global commit.gpgsign true
[root@svivanov1 ~]# git config --global gpg.program $(which gpg2)
```

Рис. 3.9: Автоматические подписи

Авторизуемся для настройки gh через браузер (рис. 10).

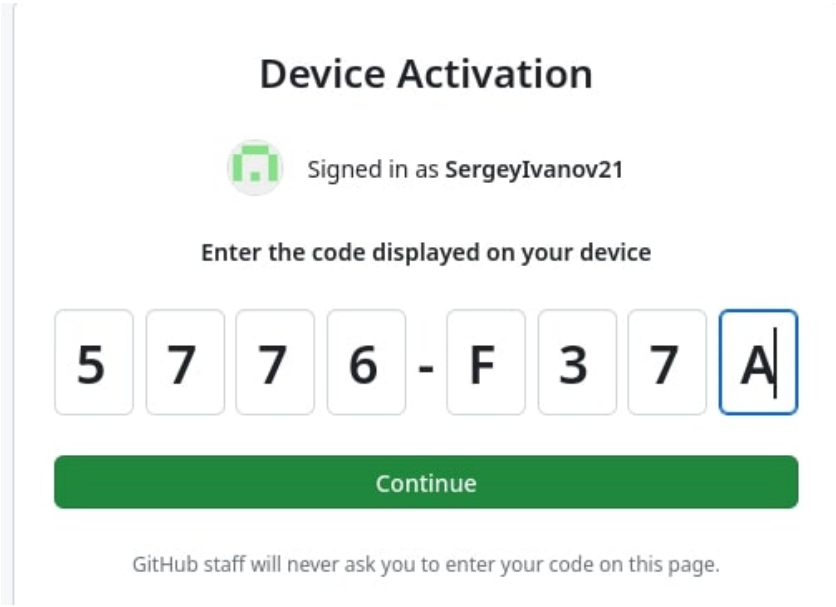

Рис. 3.10: Авторизация gh

Создадим репозиторий курса на основе шаблона и рабочее пространство (рис. 11).

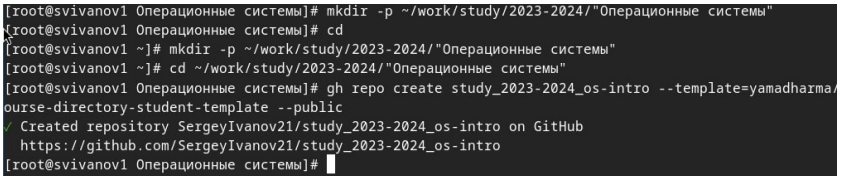

Рис. 3.11: Создание репозитория

Клонируем репозиторий в рабочее пространство (рис. 12).

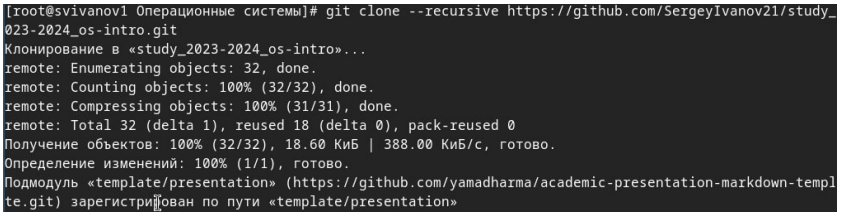

Рис. 3.12: Клонирование репозитория

Перейдем в каталог курса и удалим лишние файлы (рис. 13).

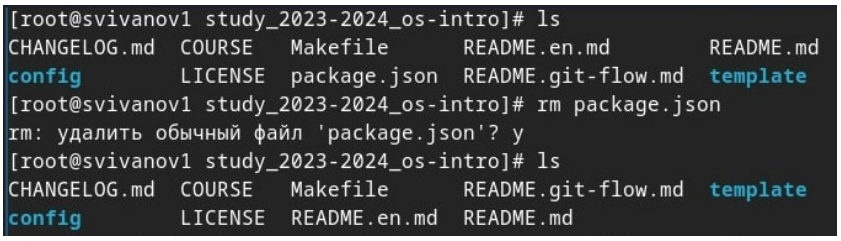

Рис. 3.13: Удаление файла

Создадим необходимые каталоги ‘echo os-intro > COURSE’ (рис. 14).

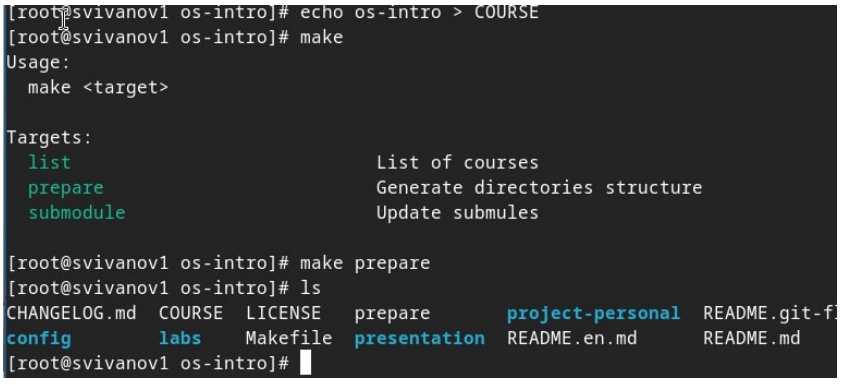

Рис. 3.14: Создание каталогов

Отправим файлы на сервер ‘git add’ (рис. 15).

```
[root@svivanov1 os-intro]# git commit -am'feat(main): make course structure'
[master f8e42e0] feat(main): make course structure
 361 files changed, 98413 insertions(+), 14 deletions(-)
 create mode 100644 labs/README.md
 create mode 100644 labs/README.ru.md
 create mode 100644 labs/lab01/presentation/Makefile
 create mode 100644 labs/lab01/presentation/image/kulyabov.jpg
 create mode 100644 labs/lab01/presentation/presentation.md
 create mode 100644 labs/lab01/report/Makefile
 create mode 100644 labs/lab01/report/bib/cite.bib
 create mode 100644 labs/lab01/report/image/placeimg_800_600_tech.jpg
```

Рис. 3.15: git add .

Добавляем комментарий и отправляем файлы (рис. 16).

```
[root@svivanov1 os-intro]# git push
Перечисление объектов: 40, готово.
Подсчет объектов: 100% (40/40), готово.
При сжатии изменений используется до 4 потоков
Сжатие объектов: 100% (30/30), готово.
Запись объектов: 100% (38/38), 342.14 КиБ | 21.38 МиБ/с, готово.
Всего 38 (изменений 4), повторно использовано 0 (изменений 0), повторно использовано пакето
remote: Resolving deltas: 100% (4/4), completed with 1 local object.
To https://github.com/SergeyIvanov21/study_2023-2024_os-intro.git
   048c7a1..f8e42e0  master -> master
[root@svivanov1 os-intro]#
```

Рис. 3.16: Отправка на сервер

4 Контрольные вопросы

1. Что такое системы контроля версий (VCS) и для решения каких задач они предназначаются? Это программное обеспечение для облегчения работы с изменяющейся информацией. 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. Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия. Хранилище (repository), или репозиторий, — место хранения всех версий и служебной информации. Commit («[трудовой] вклад») — синоним версии; процесс создания новой версии. 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. Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида. Централизованные VCS: одно основное хранилище всего проекта и каждый пользователь копирует себе необходимые ему файлы из этого репозитория, изменяет и, затем, добавляет свои изменения обратно. Децентрализованные VCS: у каждого пользователя свой вариант (возможно не один) репозитория.

4. Опишите действия с VCS при единоличной работе с хранилищем. В случае индивидуальной работы с хранилищем VCS пользователь создает, сохраняет и обновляет версии документов непосредственно в своем локальном хранилище.

5. Опишите порядок работы с общим хранилищем VCS. При работе с общим хранилищем VCS пользователи сначала загружают, обновлюи и обмениваются версиями документов с помощью коммитов и пушей.

6. Каковы основные задачи, решаемые инструментальным средством git? 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. git –version (Проверка версии Git) git init (Инициализировать ваш текущий рабочий каталог как Git-репозиторий) git clone –recursive https://www.github.com/username/reponame (Скопировать существующий удаленный Git-репозиторий) git remote (Просмотреть список текущих удалённых репозиториев Git) git remote -v (Для более подробного вывода) git add my_script.py (Можете указать в команде конкретный файл). git add . (Позволяет охватить все файлы в текущем каталоге, включая файлы, чье имя начинается с точки) git commit -am “Commit message” (Вы можете сжать все индексированные файлы и отправить коммит). git branch (Просмотреть список текущих веток можно с помощью команды branch) git –help (Чтобы узнать больше обо всех доступных параметрах и командах) git push origin master (Передать локальные коммиты в ветку удаленного репозитория).

8. Приведите примеры использования при работе с локальным и удалённым репозиториями. Пример использования локального репозитория: пользователь вносит изменения в код и сохраняет их локально. Пример использования удаленного репозитория: пользователь скачивает код с удаленного сервера, вносит изменения и отправляет их обратно на сервер.

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

ментов.

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.

5 Выводы

В результате выполнения лабораторной работы мы изучили идеологию и применение средств контроля версий а также освоили умения по работе с git.