Отчёт по лабораторной работе №2

Операционные системы

Мурашов Иван Вячеславович

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Целью данной лабораторной работы является изучение идеологии, применение средств контроля версий и освоение умений по работе с git.

2 Задание

1. Установить программное обеспечение
2. Создать базовую конфигурацию для работы с git
3. Создать ключ SSH
4. Создать ключ PGP
5. Создать профиль на GitHub и авторизоваться на устройстве
6. Создать репозиторий курса на основе шаблона
7. Настроить каталог курса

3 Теоретическое введение

Целью данной лабораторной работы является изучение идеологии, применение средств контроля версий и освоение умений по работе с git. 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.

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

4.1 Установка программного обеспечения

Для начала я устанавливаю git (рис. 4.1).

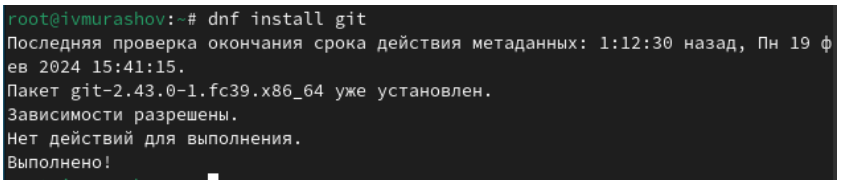

Рис. 4.1: Установка git

Затем устанавливаю gh (рис. 4.2).

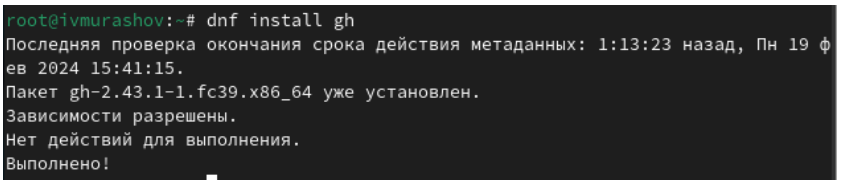

Рис. 4.2: Установка gh

4.2 Базовая настройка git

Задаю имя и email владельца репозитория (рис. 4.3).

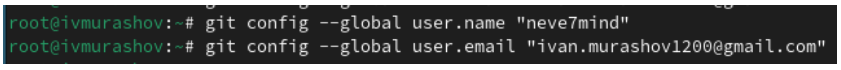

Рис. 4.3: Задание имени и email

Настравиваю utf-8 в выводе сообщений git и задаю имя начальной ветки (рис. 4.4).

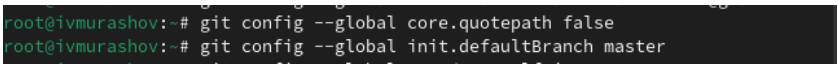

Рис. 4.4: Настройка utf-8

Настраиваю параметры autocrlf и safecrlf (рис. 4.5).

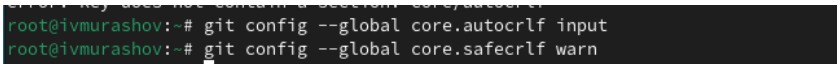

Рис. 4.5: Настройка autocrlf и safecrlf

4.3 Создание ключей

Создаю ключ SSH по алгоритму rsa с ключом размером 4096 бит (рис. 4.6).

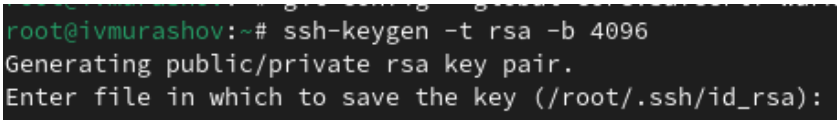

Рис. 4.6: Создание ssh ключа

А затем по алгоритму ed25519 (рис. 4.7).

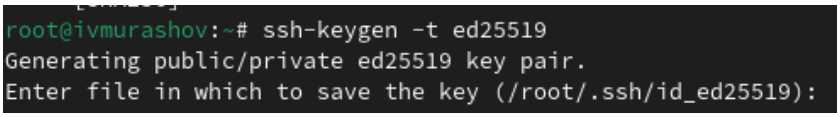

Рис. 4.7: Создание ssh ключа

Генерирую ключ PGP, выбирая из предложенных опций тип ‘RSA and RSA’, размер 4096, без срока действия. Указываю имя и адрес email (рис. 4.8).

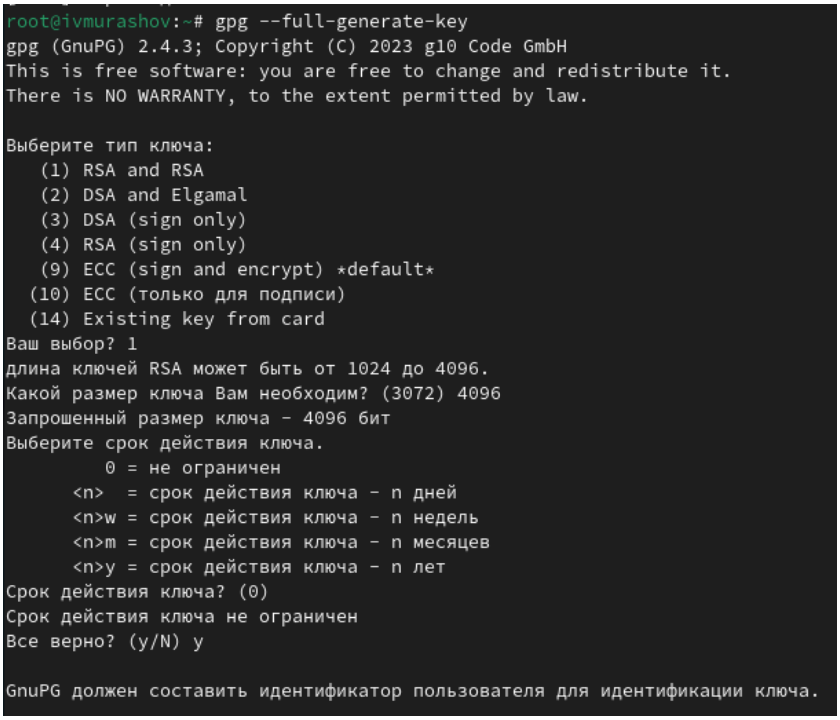

Рис. 4.8: Создание pgp ключа

4.4 Настройка github и добавление ключей

У меня уже был создан профиль на github (рис. 4.9).

Рис. 4.9: Профиль github

Вывожу список ключей (рис. 4.10).

```
root@ivmurashov:~# gpg --list-secret-keys --keyid-format LONG
gpg: проверка таблицы доверия
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: глубина: 0  достоверных:   1  подписанных:   0  доверие: 0-, 0q, 0n, 0m, 0f
, 1u
[keyboxd]
---------
sec   rsa4096/14A44B0AAB5809A0 2024-02-19 [SC]
      51EDBD9F85C58DBEF512FB1314A44B0AAB5809A0
uid                 [  абсолютно ] neve7mind <ivan.murashov1200@gmail.com>
ssb   rsa4096/26D5F9A2513E79DF 2024-02-19 [E]
```

Рис. 4.10: Список ключей

Копирую сгенерированный pgp ключ в буфер обмена (рис. 4.11).

```
root@ivmurashov:~# gpg --armor --export 14A44B0AAB5809A0 | xclip -sel clip
```

Рис. 4.11: Копирование pgp ключа

Перехожу в настройки GitHub, нажимаю на кнопку ‘New GPG key’ и вставляю скопированный ключ в поле ввода (рис. 4.12).

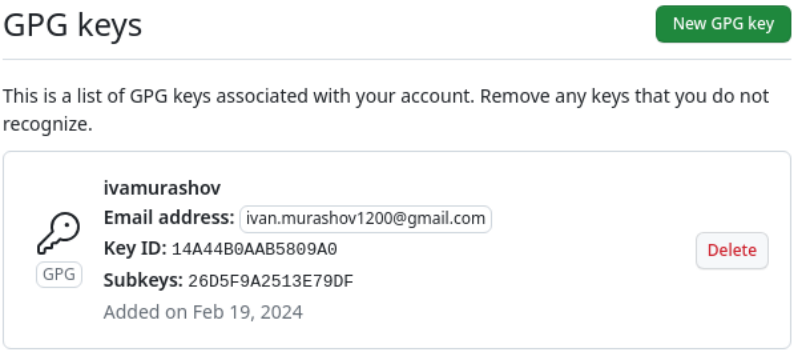

Рис. 4.12: Список ключей

4.5 Настройка автоматических подписей коммитов git и авторизация

Используя введёный email, указываю Git применять его при подписи коммитов (рис. 4.13).

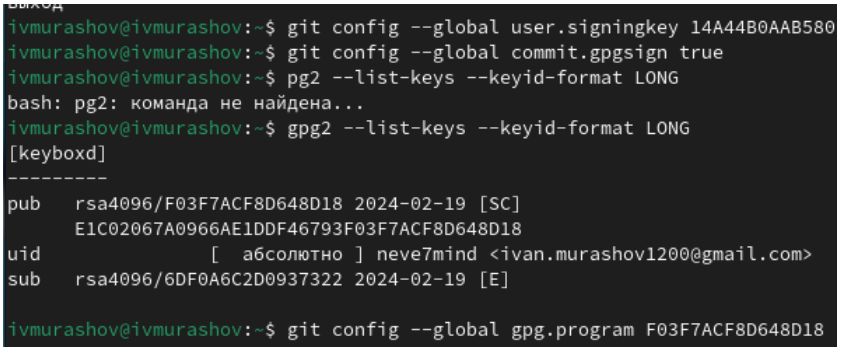

Рис. 4.13: Настройка автоматических подписей

Авторизуюсь с помощью браузера (gh login) (рис. 4.14).

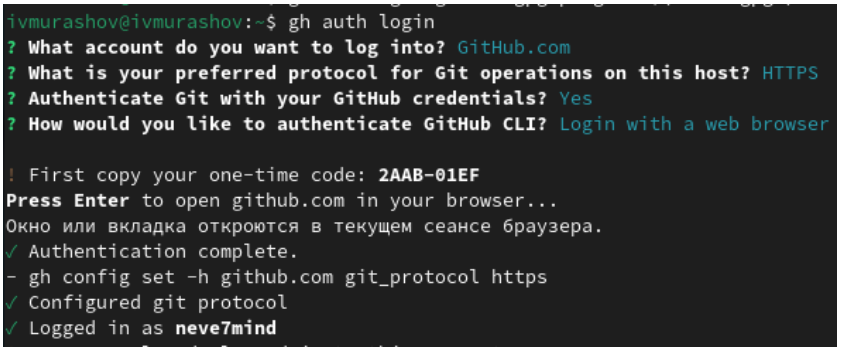

Рис. 4.14: Авторизация

4.6 Создание репозитория курса на основе шаблона и настройка каталога

Создаю каталог курса, перемещаюсь в него и создаю репозиторий на основе шаблона (рис. 4.15).

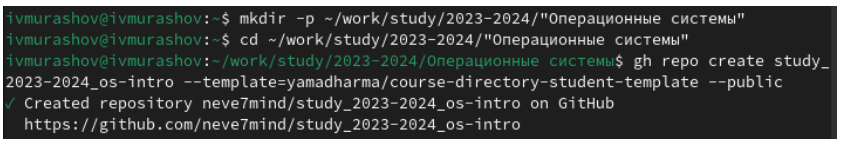

Рис. 4.15: Создание репозитория

Клонирую репозиторий (рис. 4.16).

```
ivmurashov@ivmurashov:~/work/study/2023-2024/Операционные системы$ git clone --recursive
 git@github.com:neve7mind/study_2023-2024_os-intro.git os-intro
Клонирование в «os-intro»...
remote: Enumerating objects: 32, done.
remote: Counting objects: 100% (32/32), done.
remote: Compressing objects: 100% (31/31), done.
```

Рис. 4.16: Клонирование репозитория

Удаляю лишние файлы, создаю необходимые каталоги (рис. 4.17).

```
ivmurashov@ivmurashov:~/work/study/2023-2024/Операционные системы/os-intro$ rm package.js
ivmurashov@ivmurashov:~/work/study/2023-2024/Операционные системы/os-intro$ echo os-intro > COURSE
ivmurashov@ivmurashov:~/work/study/2023-2024/Операционные системы/os-intro$ make
Usage:
  make <target>

Targets:
  list                          List of courses
  prepare                       Generate directories structure
  submodule                     Update submules

ivmurashov@ivmurashov:~/work/study/2023-2024/Операционные системы/os-intro$ make prepare
```

Рис. 4.17: Настройка каталога курса

Отправляю файлы на сервер (рис. 4.18).

```
ivmurashov@ivmurashov:~/work/study/2023-2024/Операционные системы/os-intro$ git
add .
ivmurashov@ivmurashov:~/work/study/2023-2024/Операционные системы/os-intro$ git
commit -am 'feat(main): make course structure'
Текущая ветка: master
Эта ветка соответствует «origin/master».

нечего коммитить, нет изменений в рабочем каталоге
ivmurashov@ivmurashov:~/work/study/2023-2024/Операционные системы/os-intro$ git
push
Everything up-to-date
```

Рис. 4.18: Отправка файлов на сервер

5 Ответы на контрольные вопросы

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

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.

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

Хранилище - это репозиторий, в котором хрянятся все документы, включая историю изменений.

Commit - отслеживание и сохранение изменений.

История - сохраняет в себе изменения проекта на всех его этапах.

Рабочая копия - копия проекта на основе версии из хранилища.

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

Централизованные VCS (CVS, SVN, TFS) - одно главное хранилище всего проекта, каждый участник проекта добавляет изменения в данное хранилище.

Децентрализованные VCS (Git, Bazaar) - у каждого участника свой вариант репозитория, что позволяет соединять впоследствии нужные ветки проекта.

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

Создаётся и подключается удалённый репозиторий. Участник проекта при взаимодействии с данным репозиторием отправляет все изменения на сервер.

5. Опишите порядок работы с общим хранилищем VCS.

Участник получает определённую версию проекта в хранилище. Участник отправляет все изменения на сервер. При этом старые версии проекта сохраняются.

6. Каковы основные задачи, решаемые инструментальным средством git?

Хранение информации об изменениях проекта, возможность совместной работы над одним проектом.

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 изменений: 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 <имена_
```

```
удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаёт
```

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: g

создание новой ветки, базирующейся на текущей: git checkout -b имя_ветки

переключение на некоторую ветку: git checkout имя_ветки

отправка изменений конкретной ветки в центральный репозиторий: git push origin имя_в

слияние ветки с текущим деревом: git merge --no-ff имя_ветки
```

Удаление 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.

git commit -am ‘feat(main): make course structure’ - сохранение всех добавленных изменений и всех изменённых файлов

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

Ветви - это варианты развития проекта в хранилище. Они нужны для удобства коллективной работы с хранилищем, а так же для создания “пробных” версий проекта

10. 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. Для этого сначала нужно получить список имеющихся шаблонов:

curl -L -s https://www.gitignore.io/api/list

Затем скачать шаблон, например, для C и C++

curl -L -s https://www.gitignore.io/api/c » .gitignore curl -L -s https://www.gitignore.io/api/c++ » .gitignore

6 Выводы

В ходе выполнения данной лабораторной работы я изучил идеологии и применение средств контроля версий и освоил умения по работе с git.