

Отчёт по лабораторной работе №2

Первоначальна настройка git

Седохин Даниил Алексеевич

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	12
5	Контрольные вопросы	27
6	Выводы	30

Список иллюстраций

4.1	Установка git	12
4.2	Установка gh	13
4.3	Имя владельца	14
4.4	Email владельца	14
4.5	Настройка utf-8	14
4.6	Имя начальной ветки и параметры	14
4.7	rsa	15
4.8	ed25519	16
4.9	Создание pgr	18
4.10	Копирование отпечатка приватного ключа	19
4.11	Копирование pgr ключа	20
4.12	New GPG Key	21
4.13	Подписи коммитов git	21
4.14	Авторизация	22
4.15	Создание репозитория курса	23
4.16	Завершение создания репозитория курса	24
4.17	Настройка каталога курса	25
4.18	Отправка файлов на сервер	26

1 Цель работы

Изучить идеологию и применение средств контроля версий.

Освоить умения по работе с git.

2 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.

3 Теоретическое введение

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.

Примеры использования git

Система контроля версий Git представляет собой набор программ командной строки. Благодаря тому, что Git является распределённой системой контроля версий, резервную

Основные команды git

Перечислим наиболее часто используемые команды 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
```

Стандартные процедуры работы при наличии центрального репозитория

Работа пользователя со своей веткой начинается с проверки и получения изменений из це

```
git checkout master
```

```
git pull
```

```
git checkout -b имя_ветки
```

Затем можно вносить изменения в локальном дереве и/или ветке.

После завершения внесения какого-то изменения в файлы и/или каталоги проекта необход

```
git status
```


При необходимости удаляем лишние файлы, которые не хотим отправлять в центральный репозиторий:

Затем полезно посмотреть текст изменений на предмет соответствия правилам ведения чистоты:

```
git diff
```

Если какие-либо файлы не должны попасть в коммит, то помечаем только те файлы, изменения которых мы не хотим отправлять:

```
git add ...
```

```
git rm ...
```

Если нужно сохранить все изменения в текущем каталоге, то используем:

```
git add .
```

Затем сохраняем изменения, поясняя, что было сделано:

```
git commit -am "Some commit message"
```

Отправляем изменения в центральный репозиторий:

```
git push origin имя_ветки
```

или

```
git push
```

Работа с локальным репозиторием

Создадим локальный репозиторий.

Сначала сделаем предварительную конфигурацию, указав имя и email владельца репозитория:

```
git config --global user.name "Имя Фамилия"  
git config --global user.email "work@mail"
```

Настроим utf-8 в выводе сообщений git:

```
git config --global quotepath false
```

Для инициализации локального репозитория, расположенного, например, в каталоге ~/tutorial:

```
cd  
mkdir tutorial  
cd tutorial  
git init
```

После этого в каталоге tutorial появится каталог .git, в котором будет храниться история изменений.

Создадим тестовый текстовый файл hello.txt и добавим его в локальный репозиторий:

```
echo 'hello world' > hello.txt  
git add hello.txt  
git commit -am 'Новый файл'
```

Воспользуемся командой status для просмотра изменений в рабочем каталоге, сделанных:

```
git status
```

Во время работы над проектом так или иначе могут создаваться файлы, которые не требуются для работы.

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

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

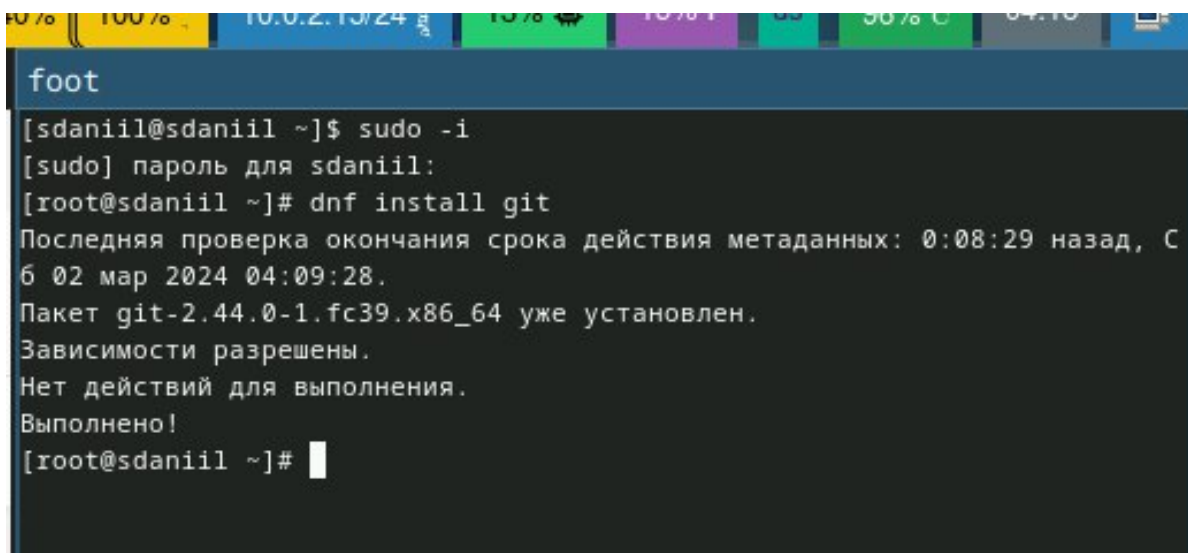
```
curl -L -s https://www.gitignore.io/api/c >> .gitignore
```

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

4 Выполнение лабораторной работы

1) Установим git:

`dnf install git` (рис. 4.1).

A screenshot of a terminal window titled 'foot'. The user 'sdaniil' is at the prompt '~]'. They enter 'sudo -i' to become root. The prompt changes to '[sudo] пароль для sdaniil:'. After entering the password, the prompt becomes '[root@sdaniil ~]#'. They then enter 'dnf install git'. The terminal shows the following output: 'Последняя проверка окончания срока действия метаданных: 0:08:29 назад, С 6 02 мар 2024 04:09:28.', 'Пакет git-2.44.0-1.fc39.x86_64 уже установлен.', 'Зависимости разрешены.', 'Нет действий для выполнения.', 'Выполнено!', and finally '[root@sdaniil ~]#' with a cursor.

```
foot
[sdaniil@sdaniil ~]$ sudo -i
[sudo] пароль для sdaniil:
[root@sdaniil ~]# dnf install git
Последняя проверка окончания срока действия метаданных: 0:08:29 назад, С
6 02 мар 2024 04:09:28.
Пакет git-2.44.0-1.fc39.x86_64 уже установлен.
Зависимости разрешены.
Нет действий для выполнения.
Выполнено!
[root@sdaniil ~]#
```

Рис. 4.1: Установка git

2) Установка gh

Fedora:

`dnf install gh` (рис. 4.2).

```

Выполнено!
[root@sdaniil ~]# dnf install gh
Последняя проверка окончания срока действия метаданных: 0:08:49 назад, С
6 02 мар 2024 04:09:28.
Зависимости разрешены.
=====
Пакет      Архитектура  Версия                Репозиторий    Размер
=====
Установка:
gh          x86_64       2.43.1-1.fc39         updates        9.1 М
=====
Результат транзакции
=====
Установка 1 Пакет

Объем загрузки: 9.1 М
Объем изменений: 46 М
Продолжить? [д/Н]: д
Загрузка пакетов:
gh-2.43.1-1.fc39.x86_64.rpm          6.1 MB/s | 9.1 MB      00:01
-----
Общий размер                        4.1 MB/s | 9.1 MB      00:02
Проверка транзакции
Проверка транзакции успешно завершена.
Идет проверка транзакции
Тест транзакции проведен успешно.
Выполнение транзакции
Подготовка      :                               1/1
Установка       : gh-2.43.1-1.fc39.x86_64    1/1
Запуск скриптлета: gh-2.43.1-1.fc39.x86_64    1/1
Проверка        : gh-2.43.1-1.fc39.x86_64    1/1

Установлен:
gh-2.43.1-1.fc39.x86_64

Выполнено!
[root@sdaniil ~]#

```

Рис. 4.2: Установка gh

3) Зададим имя и email владельца репозитория:

git config --global user.name "Name Surname"

git config --global user.email "work@mail". (рис. 4.3 4.4).

```
Выполнено!  
[root@sdaniil ~]# git config --global user.name "Sedokhin Daniil"  
[root@sdaniil ~]#
```

Рис. 4.3: Имя владельца

```
[sdaniil@sdaniil os-intro]$ git config --global user.email "sedokhin.dan  
iil@gmail.com"  
[sdaniil@sdaniil os-intro]$
```

Рис. 4.4: Email владельца

4) Настроим utf-8 в выводе сообщений git:

git config --global core.quotePath false (рис. 4.5)

```
root@sdaniil ~]# git config --global core.quotePath false  
root@sdaniil ~]#
```

Рис. 4.5: Настройка utf-8

5) Зададим имя начальной ветки (будем называть её master):

git config --global init.defaultBranch master

Также зададим следующие параметры: (рис. 4.6). Параметр autocrlf:

git config --global core.autocrlf input

Параметр safecrlf:

git config --global core.safecrlf warn

```
[root@sdaniil ~]# git config --global init.defaultBranch master  
[root@sdaniil ~]# git config --global core.autocrlf input  
[root@sdaniil ~]# git config --global core.safecrlf warn  
[root@sdaniil ~]#
```

Рис. 4.6: Имя начальной ветки и параметры

6) Создадим ключи ssh

По алгоритму rsa с ключём размером 4096 бит:

ssh-keygen -t rsa -b 4096 (рис. 4.7).

```
[root@sdaniil ~]# ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa
Your public key has been saved in /root/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:bbau3Rp9vc+YGboH6QcWZafyggmU2FUS0VHr1lDSvuA root@sdaniil
The key's randomart image is:
+---[RSA 4096]---+
|      o o==o+o. |
|      . +   .. =o. |
|      .       +oo |
|      ..  ooo.. |
|      S.+o.=o.o |
|      ooo*E.+ |
|      oo.+o . |
|      o o.oo*. |
|      ..+.++++. |
+-----[SHA256]-----+
[root@sdaniil ~]#
```

Рис. 4.7: rsa

7) Выполним те же действия по алгоритму ed25519:

ssh-keygen -t ed25519 (рис. 4.8).

```

[root@sdaniil ~]# ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/root/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_ed25519
Your public key has been saved in /root/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:6pBUex5gxmVYWJFuWgQHaAwbVK7z64jwC3rHvEW8HzE root@sdaniil
The key's randomart image is:
+--[ED25519 256]--+
| .+...oBBo      |
|  ++.o+o        |
| ... *o         |
| . = o+         |
| o . ++E        |
| + o.= +        |
|o  o+ + o       |
|o+..+= . .     |
|o.+++.. .      |
+-----[SHA256]-----+
[root@sdaniil ~]#

```

Рис. 4.8: ed25519

8) Создадим ключи pgr.

Генерируем ключ

`gpg --full-generate-key`

Из предложенных опций выбираем:

тип RSA and RSA;

размер 4096;

выберем срок действия;

значение по умолчанию — 0 (срок действия не истекает никогда).

GPG запросит личную информацию, которая сохранится в ключе:

Имя (не менее 5 символов).

Адрес электронной почты.

При вводе email убедимся, что он соответствует адресу, используемому на GitHub.

Комментарий. Можно ввести что угодно или нажать клавишу ввода, чтобы оставить это поле пустым. (рис. 4.9).

```

foot
Какой размер ключа Вам необходим? (3072) 4096
Запрошенный размер ключа - 4096 бит
Выберите срок действия ключа.
    0 = не ограничен
    <n> = срок действия ключа - n дней
    <n>w = срок действия ключа - n недель
    <n>m = срок действия ключа - n месяцев
    <n>y = срок действия ключа - n лет
Срок действия ключа? (0) 0
Срок действия ключа не ограничен
Все верно? (y/N) y

GnuPG должен составить идентификатор пользователя для идентификации ключа.

Ваше полное имя: Daniil Sedokhin
Адрес электронной почты: sedokhin.daniil@gmail.com
Примечание:
Вы выбрали следующий идентификатор пользователя:
    "Daniil Sedokhin <sedokhin.daniil@gmail.com>"

Сменить (N)Имя, (C)Примечание, (E)Адрес; (O)Принять/(Q)Выход? o
Необходимо получить много случайных чисел. Желательно, чтобы Вы
в процессе генерации выполняли какие-то другие действия (печать
на клавиатуре, движения мыши, обращения к дискам); это даст генератору
случайных чисел больше возможностей получить достаточное количество энтропии.
Необходимо получить много случайных чисел. Желательно, чтобы Вы
в процессе генерации выполняли какие-то другие действия (печать
на клавиатуре, движения мыши, обращения к дискам); это даст генератору
случайных чисел больше возможностей получить достаточное количество энтропии.
gpg: /home/sdaniil/.gnupg/trustdb.gpg: создана таблица доверия
gpg: создан каталог '/home/sdaniil/.gnupg/openpgp-revocs.d'
gpg: сертификат отзыва записан в '/home/sdaniil/.gnupg/openpgp-revocs.d/
C5447E3949940394C2EC2051092AA2CA3BA1694E.rev'.
открытый и секретный ключи созданы и подписаны.

pub   rsa4096 2024-03-02 [SC]
       C5447E3949940394C2EC2051092AA2CA3BA1694E
uid           Daniil Sedokhin <sedokhin.daniil@gmail.com>
sub   rsa4096 2024-03-02 [E]

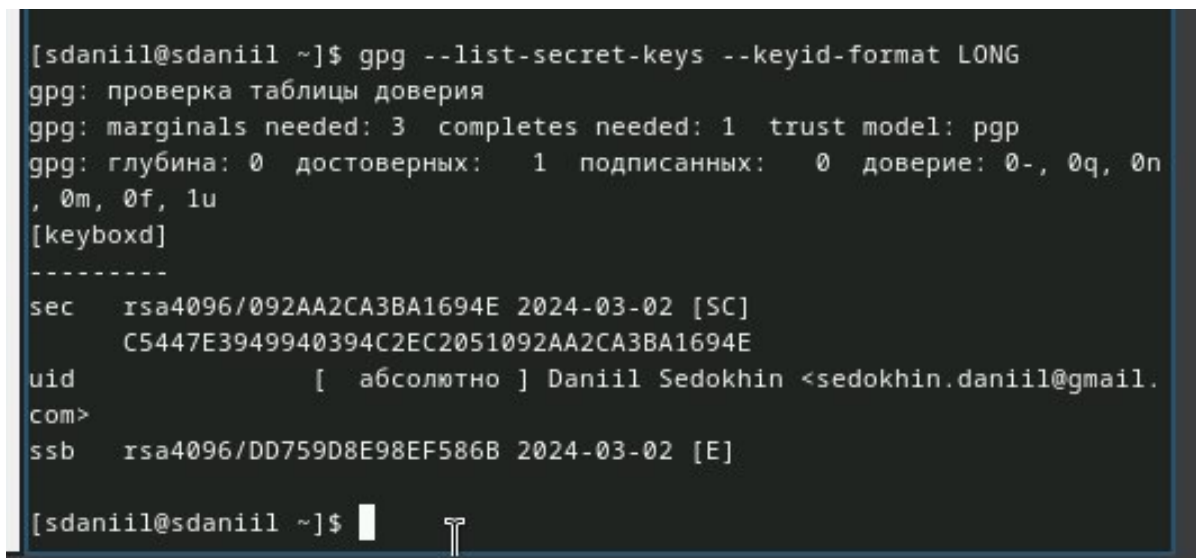
[sdaniil@sdaniil ~]$ 

```

Рис. 4.9: Создание pgp

9) Выводим список ключей и копируем отпечаток приватного ключа:

`gpg --list-secret-keys --keyid-format LONG` (рис. 4.10).



```
[sdaniil@sdaniil ~]$ 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   rsa4096/092AA2CA3BA1694E 2024-03-02 [SC]
      C5447E3949940394C2EC2051092AA2CA3BA1694E
uid           [ абсолютно ] Daniil Sedokhin <sedokhin.daniil@gmail.
com>
ssb   rsa4096/DD759D8E98EF586B 2024-03-02 [E]

[sdaniil@sdaniil ~]$
```

Рис. 4.10: Копирование отпечатка приавтного ключа

10) Скопируем сгенерированный PGP ключ в буфер обмена:

`gpg --armor --export | xclip -sel clip` (рис. 4.11).

```

gpg: filter_flush failed on close: обрыв канала
[sdaniil@sdaniil ~]$ sudo dnf install xclip
[sudo] пароль для sdaniil:
Последняя проверка окончания срока действия метаданных: 0:34:16 назад, С
б 02 мар 2024 04:09:28.
Зависимости разрешены.
=====
Пакет      Архитектура Версия                      Репозиторий Размер
=====
Установка:
xclip      x86_64      0.13-20.git11cba61.fc39          fedora        37 k
=====
Результат транзакции
=====
Установка 1 Пакет

Объем загрузки: 37 k
Объем изменений: 62 k
Продолжить? [д/Н]: д
Загрузка пакетов:
xclip-0.13-20.git11cba61.fc39.x86_64.rpm 467 kB/s | 37 kB    00:00
-----
Общий размер                      71 kB/s | 37 kB    00:00
Проверка транзакции
Проверка транзакции успешно завершена.
Идет проверка транзакции
Тест транзакции проведен успешно.
Выполнение транзакции
Подготовка      :                               1/1
Установка       : xclip-0.13-20.git11cba61.fc39.x86_64 1/1
Запуск скрипта  : xclip-0.13-20.git11cba61.fc39.x86_64 1/1
Проверка        : xclip-0.13-20.git11cba61.fc39.x86_64 1/1

Установлен:
xclip-0.13-20.git11cba61.fc39.x86_64

Выполнено!
[sdaniil@sdaniil ~]$ gpg --armor --export sedokhin.daniil@gmail.com | xc
lip -sel clip
[sdaniil@sdaniil ~]$

```

Рис. 4.11: Копирование pgp ключа

- 11) Перейдем в настройки GitHub (<https://github.com/settings/keys>), нажмем на кнопку New GPG key и вставим полученный ключ в поле ввода. (рис. 4.12).

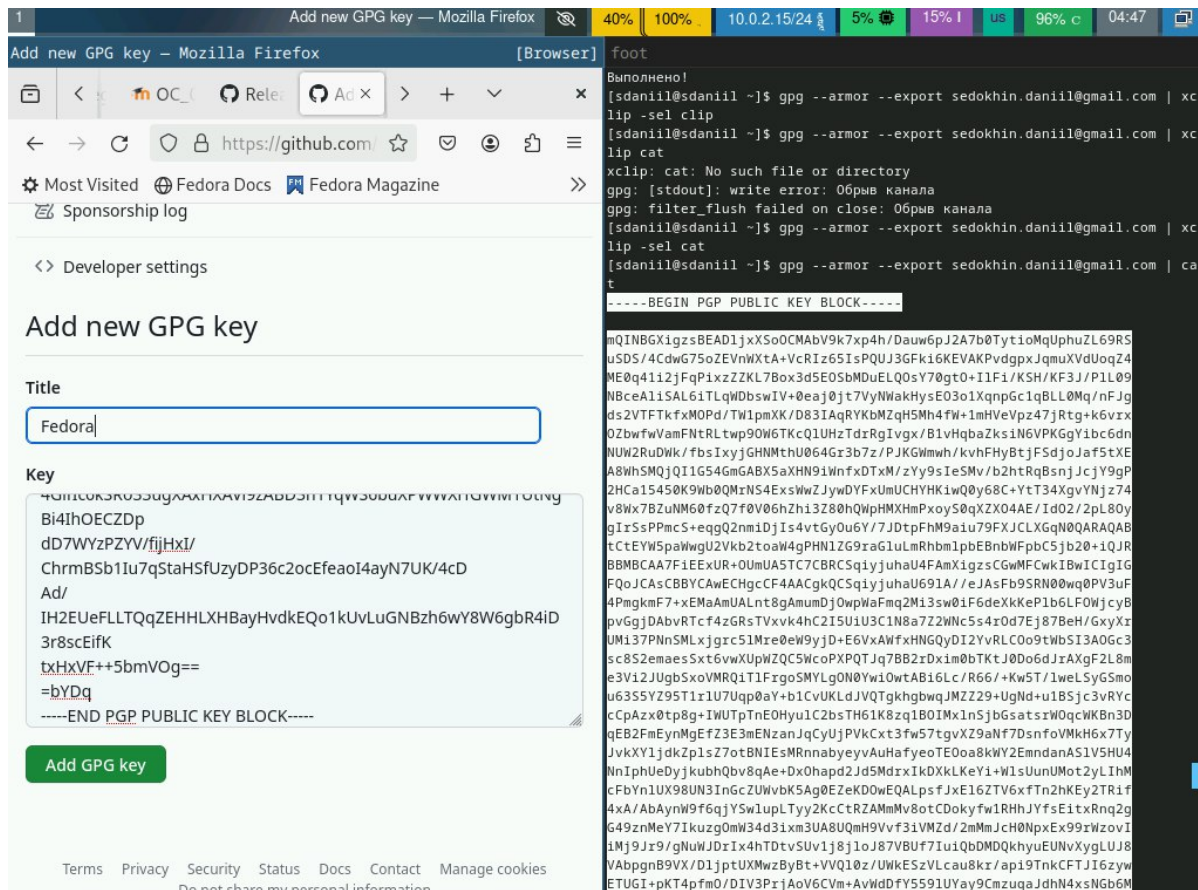


Рис. 4.12: New GPG Key

12) Настроим автоматические подписи коммитов git

Используя введённый email, укажем Git применять его при подписи коммитов:

```
git config --global user.signingkey
```

```
git config --global commit.gpgsign true
```

```
git config --global gpg.program $(which gpg2) (рис. 4.13).
```

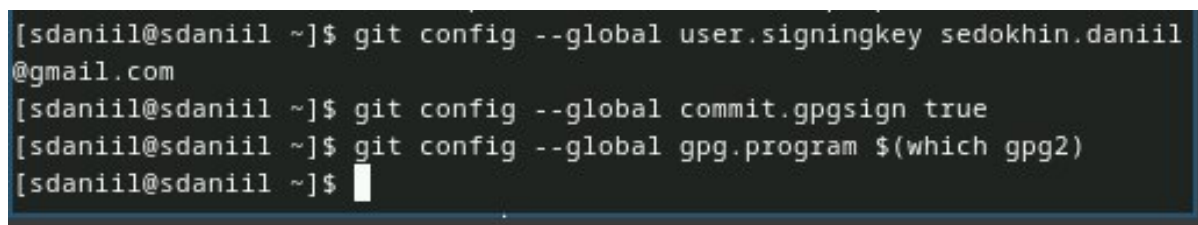


Рис. 4.13: Подписи коммитов git

13) Настройка gh

Для начала необходимо авторизоваться

gh auth login

Утилита задаст несколько наводящих вопросов, ответив на которые мы успешно авторизируемся (рис. 4.14).

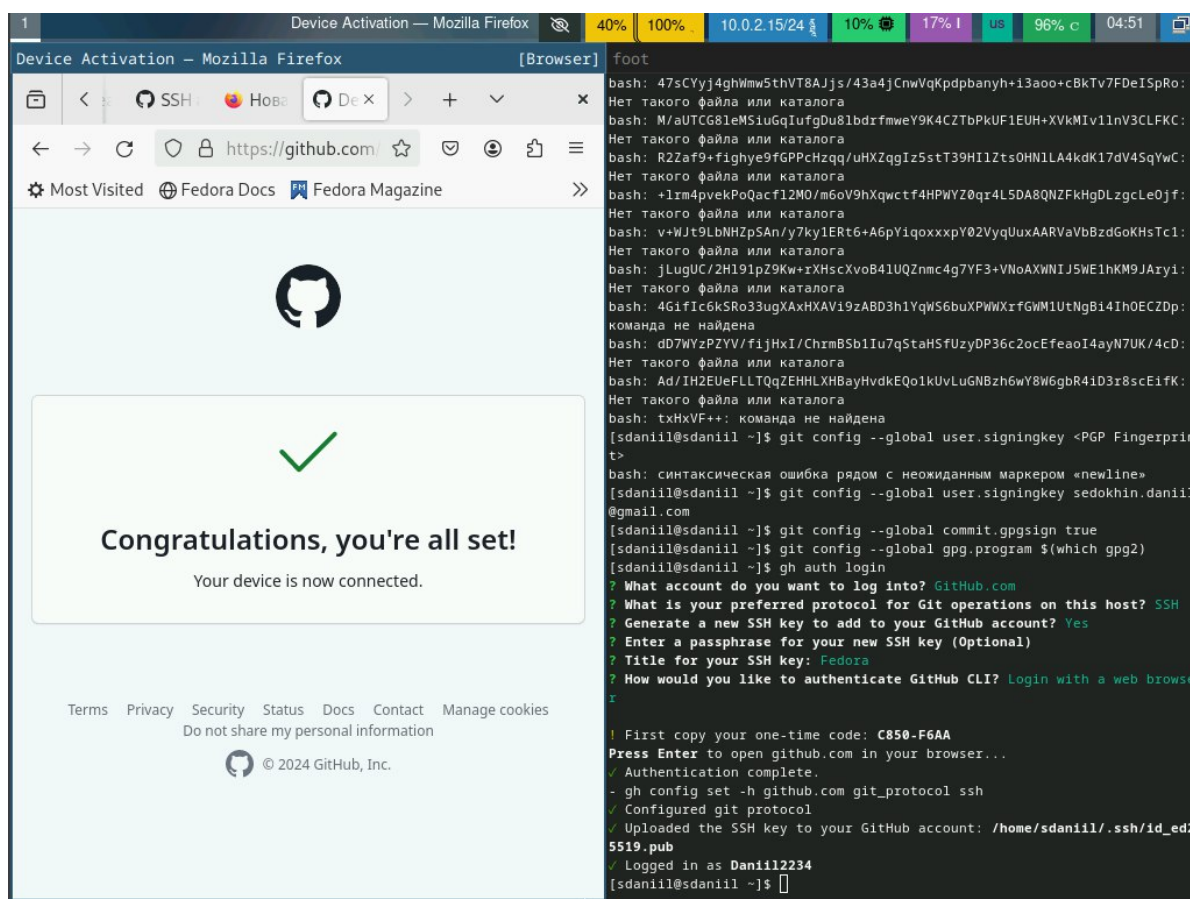


Рис. 4.14: Авторизация

14) Создание репозитория курса на основе шаблона

Необходимо создать шаблон рабочего пространства

Например, для 2023–2024 учебного года и предмета «Операционные системы» (код предмета os-intro) создание репозитория примет следующий вид (рис. 4.15) (рис. 4.16).

```

[sdaniil@sdaniil ~]$ mkdir -p ~/work/study/2023-2024/"Операционные системы"
[sdaniil@sdaniil ~]$ cd ~/work/study/2023-2024/"Операционные системы"
[sdaniil@sdaniil Операционные системы]$ gh repo create study_2023-2024_os-intro --template=yamadharm/course-directory-student-template --public

✓ Created repository Daniil2234/study_2023-2024_os-intro on GitHub
https://github.com/Daniil2234/study_2023-2024_os-intro
[sdaniil@sdaniil Операционные системы]$
[sdaniil@sdaniil Операционные системы]$ git clone --recursive git@github.com:<owner>/study_2023-2024_os-intro.git os-intro
bash: owner: Нет такого файла или каталога
[sdaniil@sdaniil Операционные системы]$ git clone --recursive git@github.com:Daniil2234/study_2023-2024_os-intro.git os-intro
Клонирование в «os-intro»...
The authenticity of host 'github.com (140.82.121.3)' can't be established.
ED25519 key fingerprint is SHA256:+DiY3wvV6TuJJhbpZisF/zLDA0zPMSvHdkr4UvC0qU.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'github.com' (ED25519) to the list of known hosts.
remote: Enumerating objects: 32, done.
remote: Counting objects: 100% (32/32), done.
remote: Compressing objects: 100% (31/31), done.
remote: Total 32 (delta 1), reused 18 (delta 0), pack-reused 0
Получение объектов: 100% (32/32), 18.59 КиБ | 18.59 МиБ/с, готово.
Определение изменений: 100% (1/1), готово.
Подмодуль «template/presentation» (https://github.com/yamadharm/academic-presentation-markdown-template.git) зарегистрирован по пути «template/presentation»
Подмодуль «template/report» (https://github.com/yamadharm/academic-laboratory-report-template.git) зарегистрирован по пути «template/report»
Клонирование в «/home/sdaniil/work/study/2023-2024/Операционные системы/os-intro/template/presentation»...
remote: Enumerating objects: 95, done.
remote: Counting objects: 100% (95/95), done.
remote: Compressing objects: 100% (67/67), done.
remote: Total 95 (delta 34), reused 87 (delta 26), pack-reused 0
Получение объектов: 100% (95/95), 96.99 КиБ | 1.05 МиБ/с, готово.

```

Рис. 4.15: Создание репозитория курса

```

Получение объектов: 100% (32/32), 18.59 КиБ | 18.59 МиБ/с, готово.
Определение изменений: 100% (1/1), готово.
Подмодуль «template/presentation» (https://github.com/yamadharm/academic-presentation-markdown-template.git) зарегистрирован по пути «template/presentation»
Подмодуль «template/report» (https://github.com/yamadharm/academic-laboratory-report-template.git) зарегистрирован по пути «template/report»
Клонирование в «/home/sdaniil/work/study/2023-2024/Операционные системы/os-intro/template/presentation»...
remote: Enumerating objects: 95, done.
remote: Counting objects: 100% (95/95), done.
remote: Compressing objects: 100% (67/67), done.
remote: Total 95 (delta 34), reused 87 (delta 26), pack-reused 0
Получение объектов: 100% (95/95), 96.99 КиБ | 1.05 МиБ/с, готово.
Определение изменений: 100% (34/34), готово.
Клонирование в «/home/sdaniil/work/study/2023-2024/Операционные системы/os-intro/template/report»...
remote: Enumerating objects: 126, done.
remote: Counting objects: 100% (126/126), done.
remote: Compressing objects: 100% (87/87), done.
remote: Total 126 (delta 52), reused 108 (delta 34), pack-reused 0

Получение объектов: 100% (126/126), 335.80 КиБ | 1.67 МиБ/с, готово.
Определение изменений: 100% (52/52), готово.
Submodule path 'template/presentation': checked out '40a1761813e197d00e8443ff1ca72c60a304f24c'
Submodule path 'template/report': checked out '7c31ab8e5dfa8cdb2d67caeb8a19ef8028ced88e'
[sdaniil@sdaniil Операционные системы]$

```

Рис. 4.16: Завершение создания репозитория курса

15) Настройка каталога курса (рис. 4.17).

Перейдем в каталог курса:

```
cd ~/work/study/2022-2023/“Операционные системы”/os-intro
```

Удалим лишние файлы:

```
rm package.json
```

Создадим необходимые каталоги:

```
echo os-intro > COURSE
```


make

```
[sdaniil@sdaniil Операционные системы]$ cd ~/work/study/2023-2024/"Опера
ционные системы"/os-intro
[sdaniil@sdaniil os-intro]$ rm package.json
[sdaniil@sdaniil os-intro]$ echo os-intro > COURSE
[sdaniil@sdaniil os-intro]$ make
Usage:
  make <target>

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

[sdaniil@sdaniil os-intro]$
```

Рис. 4.17: Настройка каталога курса

16) Отправим файлы на сервер: (рис. 4.18).

git add .

git commit -am 'feat(main): make course structure'

git push

```
create mode 100644 project-personal/stage6/report/pandoc/filters/pandoc
xnos/__init__.py
create mode 100644 project-personal/stage6/report/pandoc/filters/pandoc
xnos/core.py
create mode 100644 project-personal/stage6/report/pandoc/filters/pandoc
xnos/main.py
create mode 100644 project-personal/stage6/report/pandoc/filters/pandoc
xnos/pandocattributes.py
create mode 100644 project-personal/stage6/report/report.md
[sdaniil@sdaniil os-intro]$ git push
Перечисление объектов: 40, готово.
Подсчет объектов: 100% (40/40), готово.
При сжатии изменений используется до 6 потоков
Сжатие объектов: 100% (30/30), готово.
Запись объектов: 100% (38/38), 342.11 КиБ | 2.69 МиБ/с, готово.
Total 38 (delta 4), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (4/4), completed with 1 local object.
To github.com:Daniil2234/study_2023-2024_os-intro.git
c686b9d..a3f7ffc master -> master
[sdaniil@sdaniil os-intro]$
```

Рис. 4.18: Отправка файлов на сервер

5 Контрольные вопросы

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

Системы контроля версий (VCS) - это инструменты, которые отслеживают изменения в файловой системе с течением времени. Они предназначены для управления изменениями в коде и других файлах проекта, позволяя разработчикам работать над проектом одновременно, откатывать изменения, если что-то идет не так, и отслеживать историю изменений.

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

Хранилище (repository): Это место, где хранятся все файлы и история изменений проекта.

Commit: Это операция сохранения изменений в репозитории. При коммите фиксируются все изменения, сделанные с момента предыдущего коммита.

История (history): Это записи о всех коммитах, сделанных в репозитории.

Рабочая копия (working copy): Это копия файлов из репозитория, с которой вы работаете на вашем компьютере.

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

Централизованные VCS имеют одно основное хранилище, к которому

подключаются все клиенты.

Децентрализованные VCS позволяют каждому клиенту иметь собственное полноценное хранилище.

Примеры централизованных VCS: Subversion (SVN).

Примеры децентрализованных VCS: Git, Mercurial.

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

При единоличной работе с хранилищем в VCS вы создаете, изменяете и фиксируете изменения в рабочей копии и затем коммитите их в репозиторий.

- 5) Опишите порядок работы с общим хранилищем VCS.

Порядок работы с общим хранилищем VCS включает получение последних изменений из репозитория (pull), внесение своих изменений, фиксацию изменений (commit) и отправку их в репозиторий (push).

- 6) Каковы основные задачи, решаемые инструментальным средством git?

Основные задачи инструмента git:

Отслеживание изменений в файлах.

Управление версиями проекта.

Работа с удаленными репозиториями.

Ветвление и слияние изменений.

Работа с ветками.

- 7) Назовите и дайте краткую характеристику командам git.

Некоторые команды git:

git init: Создает новый репозиторий.

git add: Добавляет файлы в индекс для последующего коммита.

git commit: Фиксирует изменения в репозитории.

`git push`: Отправляет изменения в удаленный репозиторий.

`git pull`: Получает изменения из удаленного репозитория и объединяет их с текущей веткой.

- 8) Приведите примеры использования при работе с локальным и удалённым репозиториями.

Примеры использования при работе с локальным и удаленным репозиториями:

Локальный: Создание нового репозитория с помощью `git init`.

Удаленный: Клонирование существующего удаленного репозитория с помощью `git clone`.

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

Ветви (branches) - это параллельные линии разработки в репозитории, которые позволяют работать над разными фичами или исправлениями, не затрагивая основную ветку. Они могут быть нужны, чтобы изолировать различные функциональные изменения или исправления ошибок.

- 10) Как и зачем можно игнорировать некоторые файлы при `commit`?

Файлы могут быть проигнорированы при коммите с помощью файла `.gitignore`. В этом файле перечисляются шаблоны файлов или папок, которые не должны быть добавлены в репозиторий.

6 Выводы

Я Изучил идеологию и применение средств контроля версий.
Освоил умения по работе с git.