

# **Отчёт по лабораторной работе №3**

**Операционные системы**

Сячинова Ксения Ивановна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>7</b>
<b>4</b>	<b>Ответы на контрольные вопросы.</b>	<b>13</b>

## Список иллюстраций

3.1	Создание учётной записи . . . . .	7
3.2	Делаем конфигурацию . . . . .	7
3.3	Делаем конфигурацию . . . . .	8
3.4	Настройка git . . . . .	8
3.5	Создание ssh ключа . . . . .	9
3.6	Создание gpg ключа . . . . .	10
3.7	Добавление ключей . . . . .	10
3.8	Полученные ключи . . . . .	11
3.9	Настройка коммитов . . . . .	11
3.10	Содание репозитория . . . . .	12
3.11	Итог . . . . .	12

## **Список таблиц**

# 1 Цель работы

Цель работы: научиться оформлять отчёты с помощью легковесного языка разметки Markdown, а также познакомиться с основными возможностями разметки Markdown.

## 2 Задание

Сделайте отчёт по предыдущей лабораторной работе в формате Markdown. В качестве отчёта просьба предоставить отчёты в 3 форматах: pdf, docx и md (в архиве, поскольку он должен содержать скриншоты, Makefile и т.д.)

### 3 Выполнение лабораторной работы

1) Создаём учётную запись на github.(рис. 3.1)

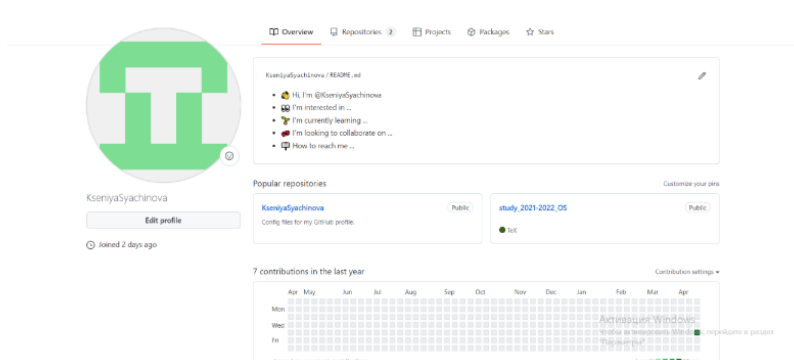


Рис. 3.1: Создание учётной записи

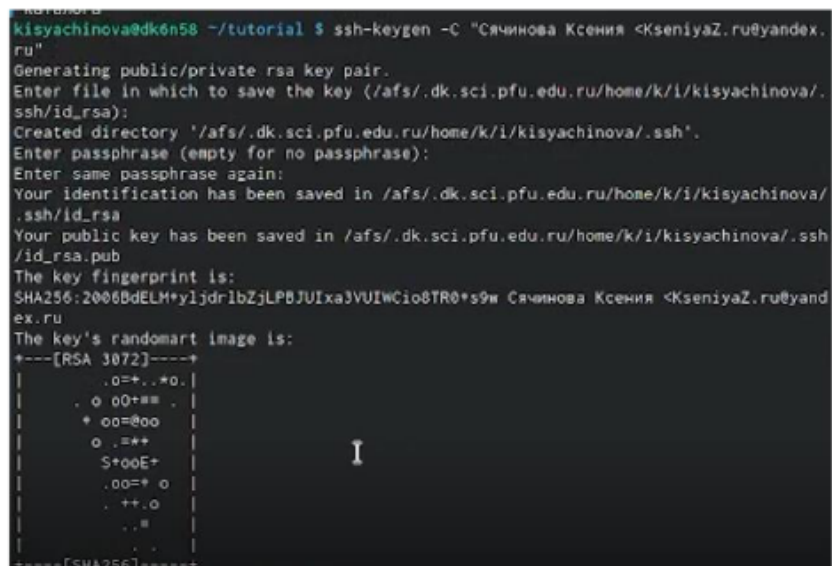
2) Сделаем предварительную конфигурацию, указав имя и email владельца репозитория с помощью `git config --global user.name "Имя Фамилия"`, `git config --global user.email "work@mail"`. (рис. 3.2)

```
kisyachinova@dk6n58 ~ $ git config --global user.name "Ksedniys Sya  
chinova"  
kisyachinova@dk6n58 ~ $  
kisyachinova@dk6n58 ~ $ git config -- global user.email "KseniyaZ.  
ru@yandex.ru"  
fatal: not in a git directory  
kisyachinova@dk6n58 ~ $ git config --global user.email "KseniyaZ.ru  
@yandex.ru"
```

Рис. 3.2: Делаем конфигурацию

3) После этого создаём новый ключ на github (команда `ssh-keygen -C "KseniyaSyachinova KseniyaZ.ru@yandex.ru"` и привязываем его к копьюте-

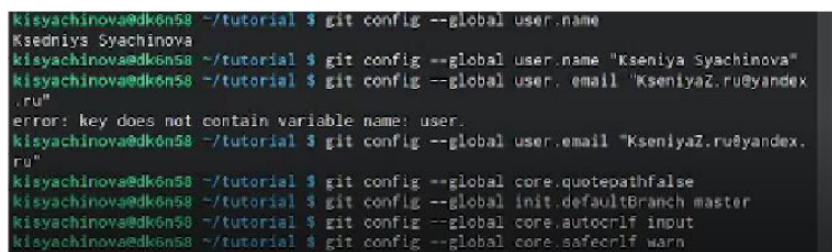
ру через консоль. После этого, скопировав из локальной консоли ключ в буфер обмена, вставляем ключ в появившееся на сайте поле. (рис. 3.3)



```
kisyachinova@dk6n58 ~/tutorial $ ssh-keygen -C "Сячинова Ксения <KseniyaZ.ru@yandex.ru>"
Generating public/private rsa key pair.
Enter file in which to save the key (/afs/.dk.sci.pfu.edu.ru/home/k/i/kisyachinova/.ssh/id_rsa):
Created directory '/afs/.dk.sci.pfu.edu.ru/home/k/i/kisyachinova/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /afs/.dk.sci.pfu.edu.ru/home/k/i/kisyachinova/.ssh/id_rsa
Your public key has been saved in /afs/.dk.sci.pfu.edu.ru/home/k/i/kisyachinova/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:2006BdELM+yldrlbZjLPBJUxa3VUIWcio8TR0*s9w Сячинова Ксения <KseniyaZ.ru@yandex.ru>
The key's randomart image is:
+---[RSA 3072]---+
|
|.o=+.o.
| o o0+=.
| + o0=@oo
| o .==+
| S+ooE+
| .oo=+ o
| . ++.o
| ..#
|
+----[SHA256]-----+
```

Рис. 3.3: Делаем конфигурацию

- 4) Приступаем к базовой настройке git. Зададим имя и email владельца репозитория: `git config --global user.name "Name Surname"`, `git config --global user.email "work@mail"`. Настроим utf-8 в выводе сообщений git: `git config --global core.quotepath false`. Настроим верификацию и подписание коммитов git. Зададим имя начальной ветки: `git config --global init.defaultBranch master`. Параметр `autocrlf`: `git config --global core.autocrlf input`. Параметр `safecrlf`: `git config --global core.safecrlf warn`. (рис. 3.4)



```
kisyachinova@dk6n58 ~/tutorial $ git config --global user.name Kseniya Syachinova
kisyachinova@dk6n58 ~/tutorial $ git config --global user.name "Kseniya Syachinova"
kisyachinova@dk6n58 ~/tutorial $ git config --global user.email "KseniyaZ.ru@yandex.ru"
error: key does not contain variable name: user.
kisyachinova@dk6n58 ~/tutorial $ git config --global user.email "KseniyaZ.ru@yandex.ru"
kisyachinova@dk6n58 ~/tutorial $ git config --global core.quotepath false
kisyachinova@dk6n58 ~/tutorial $ git config --global init.defaultBranch master
kisyachinova@dk6n58 ~/tutorial $ git config --global core.autocrlf input
kisyachinova@dk6n58 ~/tutorial $ git config --global core.safecrlf warn
```

Рис. 3.4: Настройка git



- 5) Затем создаём свой ключ по алгоритму: `ssh-keygen -t rsa -b 4096` и `ssh-keygen -t ed25519`. (рис. 3.5)

```
kisyachinova@dk6n58 ~/tutorial $ ssh-keygen -t rsa -b4096
Generating public/private rsa key pair.
Enter file in which to save the key (/afs/.dk.sci.pfu.edu.ru/home/k/i/kisyachinova/.ssh/id_rsa):
/afs/.dk.sci.pfu.edu.ru/home/k/i/kisyachinova/.ssh/id_rsa already exists.
Overwrite (y/n)?
kisyachinova@dk6n58 ~/tutorial $ ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/afs/.dk.sci.pfu.edu.ru/home/k/i/kisyachinova/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /afs/.dk.sci.pfu.edu.ru/home/k/i/kisyachinova/.ssh/id_ed25519
Your public key has been saved in /afs/.dk.sci.pfu.edu.ru/home/k/i/kisyachinova/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:5cm1e3j5b1d3fy71uiAZIyqJYbzdLNFtrkbF0Xe0W10 kisyachinova@dk6n58
The key's randomart image is:
+--[ED25519 256]--+
|
| . . . o .
| o . = . + =E.
| . =.oo +S.O.+ .
| =.* ...o..B . B|
|...+ .o o .+*+|
| ..0 .. . . . *|
| .. o*o|
+-----[SHA256]-----+
kisyachinova@dk6n58 ~/tutorial $
```

Рис. 3.5: Создание ssh ключа

- 6) Создаём ключ `gpg`: `gpg --full-generate-key`. Затем настраиваем: Тип RSA and RSA Размер 4096 Срок действия, значение по умолчанию — 0 (срок действия не истекает никогда). Имя Адрес электронной почты (рис. 3.6)

```
kisyachinova@dk6n58 ~/tutorial $ gpg --full-generate-key
gpg (GnuPG) 2.2.27; Copyright (C) 2021 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Выберите тип ключа:
  (1) RSA и RSA (по умолчанию)
  (2) DSA и Elgamal
  (3) DSA (только для подписи)
  (4) RSA (только для подписи)
  (14) Имеющийся на карте ключ
Ваш выбор? 1
длина ключей RSA может быть от 1024 до 4096.
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: Kseniya Syachinova
Адрес электронной почты: KseniyaZ.ru@yandex.ru
Примечание:
Вы выбрали следующий идентификатор пользователя:
  "Kseniya Syachinova <KseniyaZ.ru@yandex.ru>"

Сменить (N)Имя, (C)Примечание, (E)Адрес: ((O)Организация)
Необходимо получить много случайных чисел
в процессе генерации выполняли какие-то д
на клавиатуре, движения мыши, обращения к
случайных чисел больше возможностей получ
```

Рис. 3.6: Создание gpg ключа

- 7) Добавляем PGP ключ в GitHub. Используем `gpg --list-secret-keys --keyid-format LONG`. По образцу видим отпечаток моего ключа, вставляем его в следующую конструкцию: `gpg--armor--export<pgpFingerprint> | xclip-selclip`. Затем перешли в настройки github и вставили полученный ключ. (рис. 3.7)

```
kisyachinova@dk6n58 ~/tutorial $ 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
/afs/.dk.sci.pfu.edu.ru/home/k/i/kisyachinova/.gnupg/pubring.kbx
-----
sec   rsa4096/9D4621DBCE7B46BA 2022-04-21 [SC]
      7EA44CEf09A5CCBEEAD49B4F9D4621DBCE7B46BA
uid   [ абсолютно ] Kseniya
ssb   rsa4096/B8E84A634B9942CB 2022-04-21

kisyachinova@dk6n58 ~/tutorial $ gpg --armor --export 9D4621DBCE7B46BA | xclip -sel clip
```

Рис. 3.7: Добавление ключей

- 8) Таким образом, у нас получились следующие ssh и gpg ключи: (рис. 3.8)

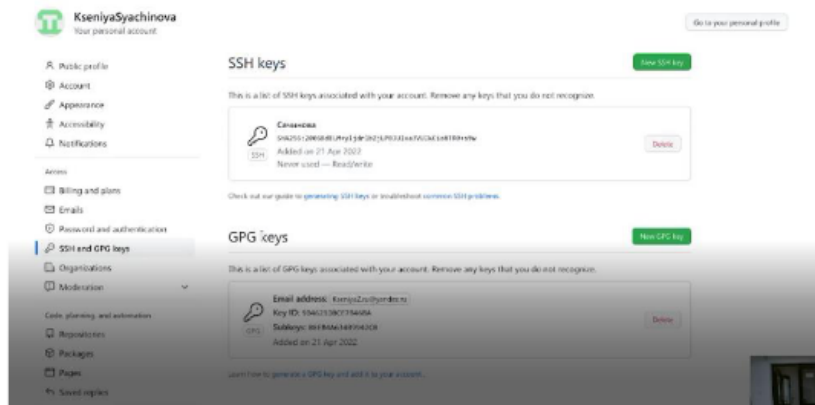


Рис. 3.8: Полученные ключи

9) Затем настраиваем автоматические подписи коммитов git: `git config --global user.signingkey`, `git config --global commit.gpgsign true`, `git config --global gpg.program $(which gpg2)` (рис. 3.9)

```
kisyachinova@dk6n58 ~/tutorial $ git config --global user.signingkey 904621DBCE7B46B
A
kisyachinova@dk6n58 ~/tutorial $ git config --global commit.gpgsigntrue
kisyachinova@dk6n58 ~/tutorial $ git config --global gpg.program$(which gpg2)
error: недействительный ключ: gpg.program/usr/bin/gpg2
kisyachinova@dk6n58 ~/tutorial $ git config --global gpg.program$(which gpg)
error: недействительный ключ: gpg.program/usr/bin/gpg
kisyachinova@dk6n58 ~/tutorial $ git config --global gpg.program $(which gpg2)
```

Рис. 3.9: Настройка коммитов

10) После этого создаём репозиторий курса на основе шаблона. (рис. 3.10)

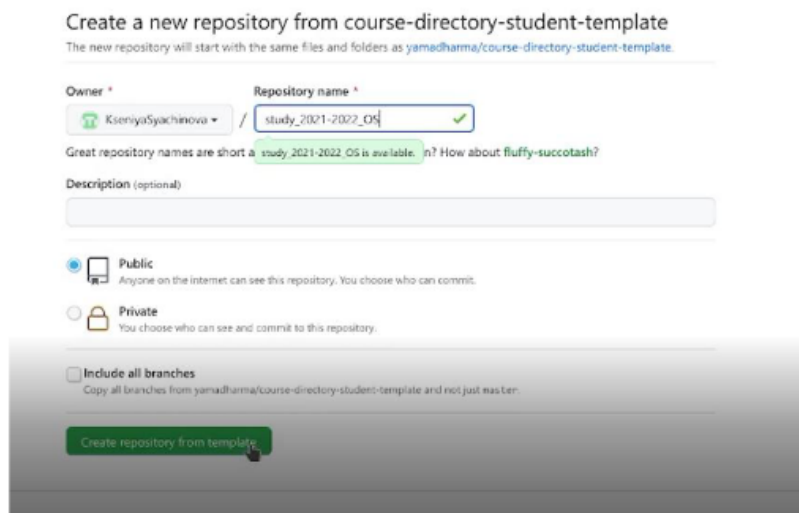


Рис. 3.10: Создание репозитория

- 11) Копируем ссылку и с помощью `git clone -recursive` добавляем наши лабораторные работы на github. Впоследствии удаляем лишние файлы: `rm package.json`, создаём необходимые каталоги: `make COURSE=os-intro`, и отправляем файлы на сервер: `git add .`, `git commit -am 'feat(main): make course structure'`, `git push`. (рис. 3.11)

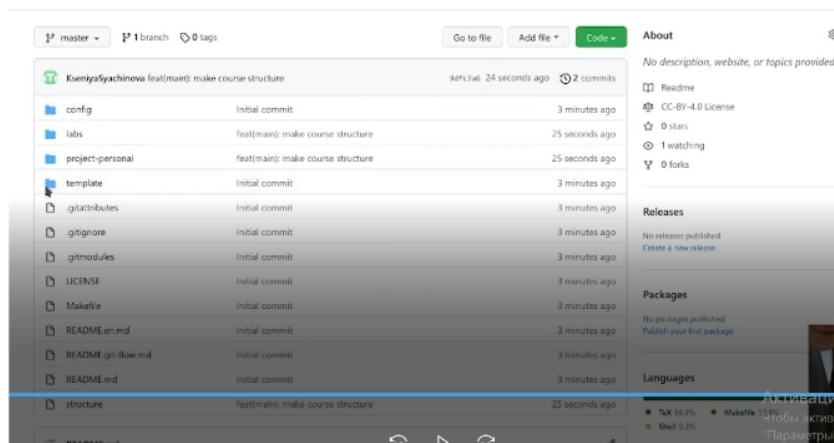


Рис. 3.11: Итог

## 4 Ответы на контрольные вопросы.

- 1). Система контроля версий Git представляет собой набор программ командной строки. Доступ к ним можно получить из терминала посредством ввода команд `git` различными опциями. Системы контроля версий (Version Control System, VCS) применяются при работе нескольких человек над одним проектом.
- 2). 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 неполную версию изменённых файлов, а производить так называемую дельта-компрессию — сохранять только изменения между последовательными версиями, что позволяет уменьшить объём хранимых данных. 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.
- 3). Централизованные системы — это системы, которые используют архитектуру

клиент / сервер, где один или несколько клиентских узлов напрямую подключены к центральному серверу. Пример - Wikipedia. В децентрализованных системах каждый узел принимает свое собственное решение. Конечное поведение системы является совокупностью решений отдельных узлов. Пример — Bitcoin. В классических системах контроля версий используется централизованная модель, предполагающая наличие единого репозитория для хранения файлов. Выполнение большинства функций по управлению версиями осуществляется специальным сервером.

4). 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"` и настроив utf-8 в выводе сообщений: `git config --global core.quotePath false`. Для инициализации локального репозитория, расположенного, например, в каталоге `~/tutorial`, необходимо ввести в командной строке: `cd ~/tutorial` `mkdir tutorial` `cd tutorial` `git init`

5). Для последующей идентификации пользователя на сервере репозитория необходимо сгенерировать пару ключей (приватный и открытый): `ssh-keygen -C "Имя Фамилия work@mail"`. Ключи сохраняются в каталоге `~/.ssh/`. Скопировав из локальной консоли ключ в буфер обмена: `cat ~/.ssh/id_rsa.pub | xclip -sel clip` вставляем ключ в появившееся на сайте поле.

6). 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). Основные команды git: 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` — 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_файлов` — удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локаль-

ной директории): `git rm имена_файлов` –сохранение добавленных изменений:  
–сохранить все добавленные изменения и все изменённые файлы: `git commit-am`  
‘Описание коммита’–сохранить добавленные изменения с внесением коммента-  
рия 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 кон-  
кретной ветки в центральный репозиторий: `git push origin имя_ветки`–слияние  
ветки стекущим деревом:`git merge -no-ff имя_ветки`–удаление ветки: –удаление  
локальной уже слитой с основным деревом ветки:`gitbranch -d имя_ветки`–прину-  
дительное удаление локальной ветки:`git branch -D имя_ветки`–удаление ветки  
с центрального репозитория: `git push origin :имя_ветки` 8). Исполнения `git`  
при работе локальными репозиториями (добавления текстового документа в  
локальный репозиторий):`git add hello.txtgit commit -am’Новыйфайл` 9). Пробле-  
мы, которые решают ветки `git`:· нужно постоянно создавать архивы с рабочим  
кодом сложно”переключаться” между архивами· сложно перетаскивать измене-  
ния между архивами· легко что-то напутать или потерять 10). Во время работы  
над проектом так или иначе могут создаваться файлы, которые не требуется добав-  
лять в последствии в репозиторий. Например, временные файлы, со-здаваемые  
редакторами, или объектные файлы, создаваемые компиляторами. Можно про-  
писать шаблоны игнорируемых при добавлении в репозиторий типов файлов в  
файл `.gitignore` с помощью `с ервисов`. Для этого сначала нужно получить списки  
меющихся шаблонов: `curl -L -s https://www.gitignore.io/api/list` Затем скачать шаб-  
лон, например, для C и C++ `curl -L -s https://www.gitignore.io/api/c » .gitignorecurl`  
`-L -s https://www.gitignore.io/api/c++ » .gitignore` # Выводы

Я научилась оформлять отчёты в Markdown, познакомилась с основными его  
возможностями.