

# **Отчет по лабораторной работе №2**

**Githab**

Легиньких Галина Андреевна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Выполнение лабораторной работы</b>	<b>6</b>
<b>3</b>	<b>Выводы</b>	<b>10</b>
<b>4</b>	<b>Контрольные вопросы</b>	<b>11</b>

## Список иллюстраций

2.1	Имя . . . . .	6
2.2	email . . . . .	6
2.3	git . . . . .	6
2.4	Имя ветки . . . . .	7
2.5	Параметр1 . . . . .	7
2.6	Параметр2 . . . . .	7
2.7	SSH . . . . .	7
2.8	PGP . . . . .	7
2.9	Синхронизация . . . . .	8
2.10	Подписи . . . . .	8
2.11	Подписи2 . . . . .	8
2.12	репозиторий . . . . .	8
2.13	github . . . . .	9
2.14	Каталог . . . . .	9
2.15	Мой github . . . . .	9

## Список таблиц

# 1 Цель работы

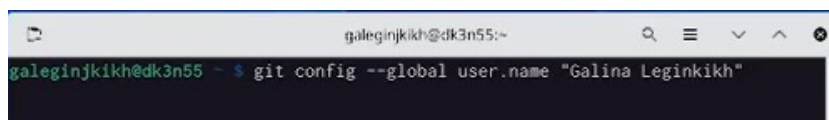
Изучить идеологию и применение средств контроля версий.

## 2 Выполнение лабораторной работы

*При выполнении лабораторной работы не записался экран монитора. Удаление учетной записи на github вызвало затруднение. Поэтому в видео описывается процесс выполнения работы после ее основного выполнения.*

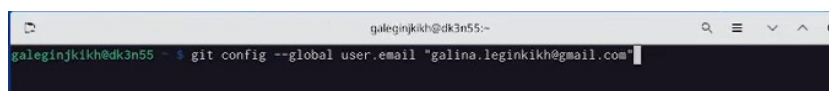
1. До выполнения лабораторной работы у меня уже была создана учетная запись на <https://github.com/>

2. Базовая настройка git: Задала имя и email владельца репозитория.(рис. 2.1)(рис. 2.2)



```
galeginkikh@dk3n55:~$ git config --global user.name "Galina Leginkikh"
```

Рис. 2.1: Имя



```
galeginkikh@dk3n55:~$ git config --global user.email "galina.leginkikh@gmail.com"
```

Рис. 2.2: email

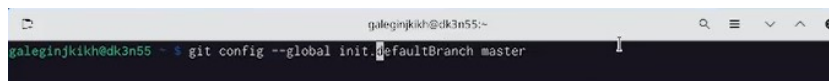
Настроила верификацию и подписание коммитов git. (рис. 2.3)



```
galeginkikh@dk3n55:~$ git config --global core.quotepath false
```

Рис. 2.3: git

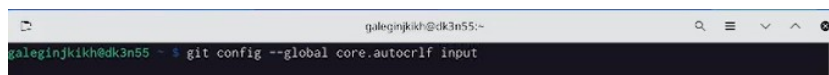
Зададим имя начальной ветки (будем называть её master).(рис. 2.4)



```
galeginkikh@dk3n55:~$ git config --global init.defaultBranch master
```

Рис. 2.4: Имя ветки

Установила параметры.(рис. 2.5)(рис. 2.6)



```
galeginkikh@dk3n55:~$ git config --global core.autocrlf input
```

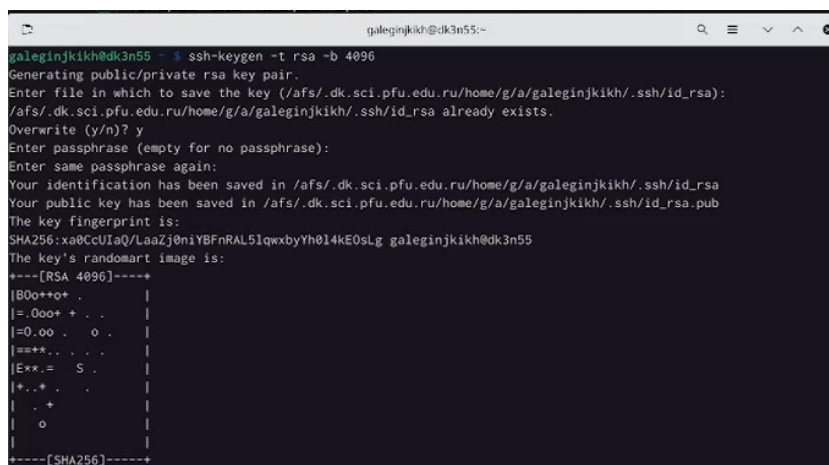
Рис. 2.5: Параметр1



```
galeginkikh@dk3n55:~$ git config --global core.safecrlf warn
```

Рис. 2.6: Параметр2

3. Запросим SSH-ключ (рис. 2.7)



```
galeginkikh@dk3n55:~$ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/afs/.dk.sci.pfu.edu.ru/home/g/a/galeginkikh/.ssh/id_rsa):
/afs/.dk.sci.pfu.edu.ru/home/g/a/galeginkikh/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /afs/.dk.sci.pfu.edu.ru/home/g/a/galeginkikh/.ssh/id_rsa
Your public key has been saved in /afs/.dk.sci.pfu.edu.ru/home/g/a/galeginkikh/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:xa0CcUIaQ/LaaZj0niYBFnRAL5lqwxbyYh014kE0sLg galeginkikh@dk3n55
The key's randomart image is:
+--[RSA 4096]-----+
| 80o++o+ . |
| =.0oo+ + . |
| =0.oo . o . |
| ==+ . . . . |
| E*+ . S . |
| +..+ . . |
| . + |
| o |
+-----[SHA256]-----+
```

Рис. 2.7: SSH

4. Командой `cat ~/.ssh/id_rsa.pub | xclip -sel clip` копируем ключ в нужном виде и вводим на сайт

5. Сгенерировала ключ PGP и добавляем его на git. (рис. 2.8)



```
galeginkikh@dk3n55:~$ pgp --full-generate-key
```

Рис. 2.8: PGP

Теперь терминал и github синхронизированы.(рис. 2.9)

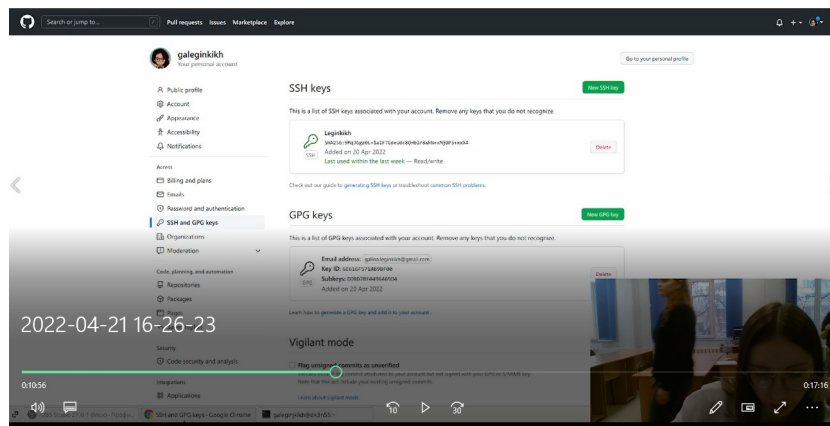


Рис. 2.9: Синхронизация

6. Настроила автоматические подписи коммитов git.(рис. 2.10)(рис. 2.11)

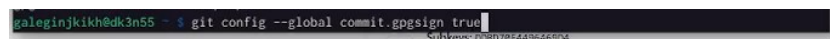


Рис. 2.10: Подписи



Рис. 2.11: Подписи2

7. Создала репозиторий курса на основе шаблона. (рис. 2.12)(рис. 2.13)

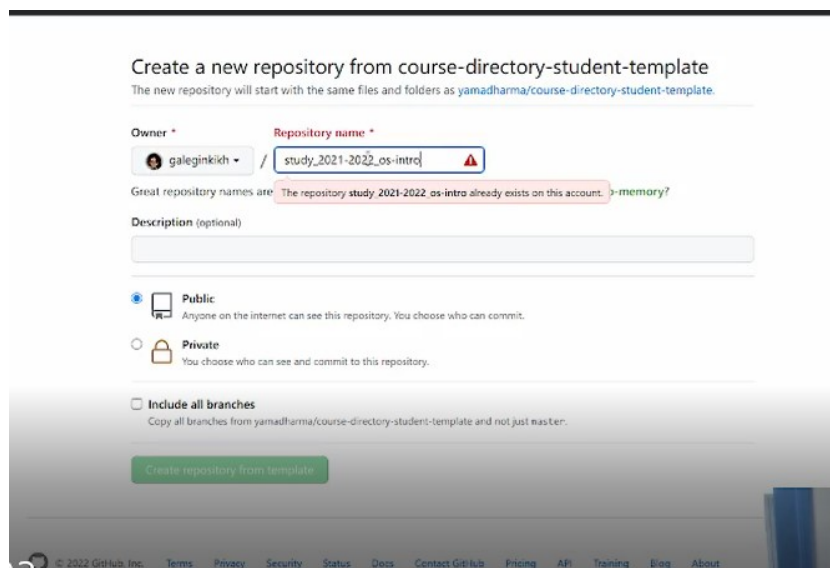


Рис. 2.12: репозиторий



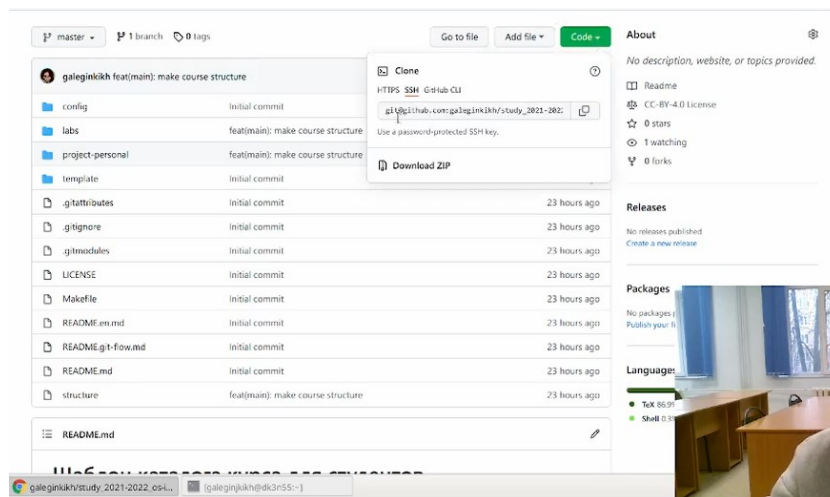


Рис. 2.13: github

8. Перешла в каталог курса.(рис. 2.14)



Рис. 2.14: Каталог

9. Удалила лишние файлы и создала необходимые каталоги.

10. Отправила файлы на сервер. (рис. 2.15) `git add .` `git commit -am 'feat(main): make course structure'` `git push`

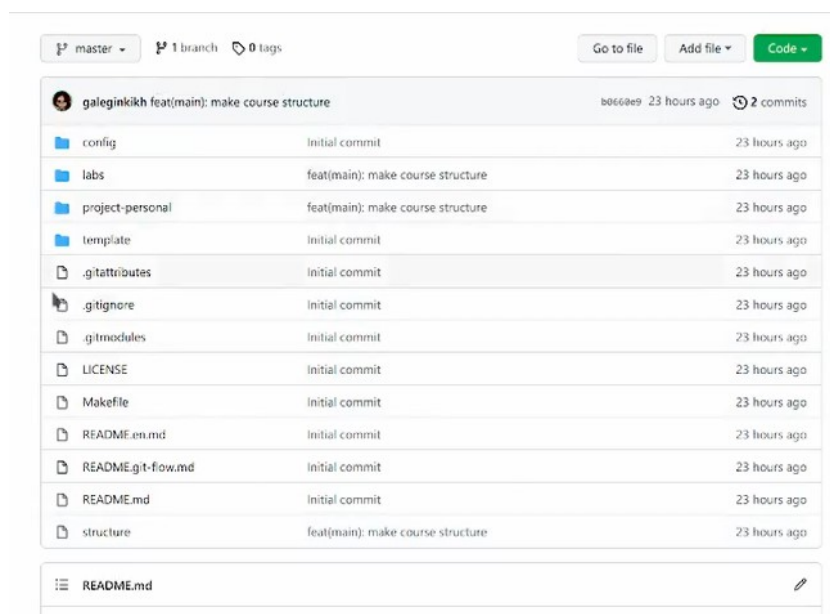


Рис. 2.15: Мой github

## 3 Выводы

Мы изучили идеологию и применение средств контроля версий. Зарегистрировались на github. Создали репозиторий группы.

## 4 Контрольные вопросы

1. Что такое системы контроля версий (VCS) и для решения каких задач они предназначены? Система контроля версий (VCS) — это место хранения кода. Система управления версиями 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. Такие системы наиболее широко используются при разработке программного обеспечения для хранения исходных кодов разрабатываемой программы.
2. Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия. Репозиторий - хранилище версий - в нем хранятся все документы вместе с историей их изменения и другой служебной информацией Commit («[трудовой] вклад», не переводится) — процесс создания новой версии Рабочая копия (working copy) — текущее состояние файлов проекта, основанное на версии, загруженной из хранилища (обычно на последней). Версия (revision), или ревизия, — состояние всех файлов на определенный момент времени, сохраненное в репозитории, с дополнительной информацией
3. Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида. 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)

4. Опишите действия с VCS при единоличной работе с хранилищем.
5. Опишите порядок работы с общим хранилищем VCS.
6. Каковы основные задачи, решаемые инструментальным средством git? У Git есть две основные задачи: хранить информацию обо всех изменениях в коде, начиная с самой первой строчки, и обеспечить удобства командной работы над кодом.
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_файлов` – удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной 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 комментария 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 конкретной ветки в центральный репозиторий: `git push origin имя_ветки` – сливание 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. Что такое и зачем могут быть нужны ветви (branches)? ‘Git branch’ – это команда для управления ветками в репозитории Git. Ветка – это просто «скользящий» указатель на один из коммитов. Когда мы создаём новые коммиты, указатель ветки автоматически сдвигается вперёд, к вновь созданному коммиту. Ветки используются для разработки одной части функционала изолированно от других. Каждая ветка представляет собой отдельную копию кода проекта. Ветки позволяют одновременно работать над разными версиями проекта. Ветвление («ветка», branch) — один из параллельных участков истории в одном хранилище, исходящих из одной версии (точки ветвления). Ветки нужны для того, чтобы программисты могли вести совместную работу над проектом и не мешать друг другу при этом.
9. Как и зачем можно игнорировать некоторые файлы при commit? Игнорируемые файлы обычно представляют собой файлы, специфичные для платформы, или автоматически созданные из сборочных систем. Временно игнорировать изменения в файле можно командой: `git update-index --assume-unchanged`