ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2

«Управление версиями»

Коняева Марина Александровна

Содержание

[Цель работы 1](#_Toc101964634)

[Теоретическое введение 1](#_Toc101964635)

[Выполнение лабораторной работы 1](#_Toc101964636)

[Выводы 8](#_Toc101964637)

[Контрольные вопросы 8](#_Toc101964638)

# Цель работы

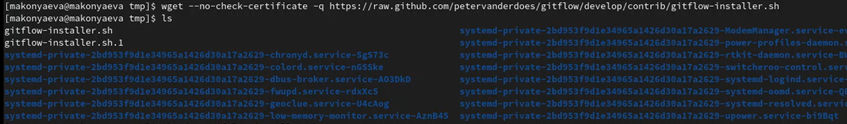
Изучить идеологию и применение средств контроля версий, освоить умения по работе с git.

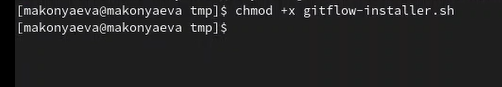
# Теоретическое введение

Markdown — язык текстовой разметки, созданный писателем и блогером Джоном Грубером. Он предназначен для создания красиво оформленных текстов в обычных файлах формата TXT. Вам не нужны громоздкие процессоры вроде Word или Pages, чтобы создавать документы с жирным или курсивным начертанием, цитатами, ссылками и даже таблицами. Достаточно запомнить простые правила Markdown, и можно писать хоть в «Блокноте». Хотя специализированные Markdown-редакторы, конечно, намного удобнее.

# Выполнение лабораторной работы

1. Заведем учетную запись на сайте github.com, свой аккаунт я создала ранее, введя необходимые данные
2. Скачивание git-flow и выдача прав на выполнение

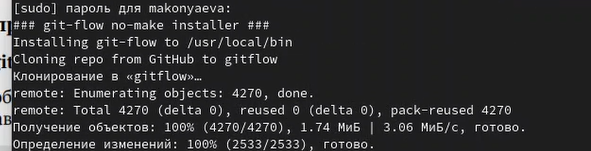
 *Изображение 2.1 Скачивание git-flow*



выдача прав на выполнение

1. Установка git-flow

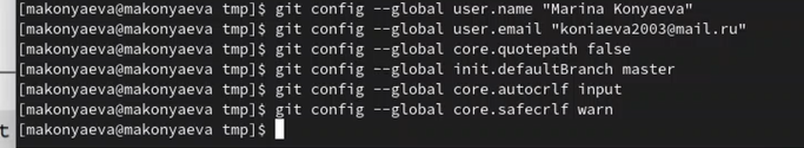
Установка git-flow *Изображение 3.1 Установка git-flow*

\*

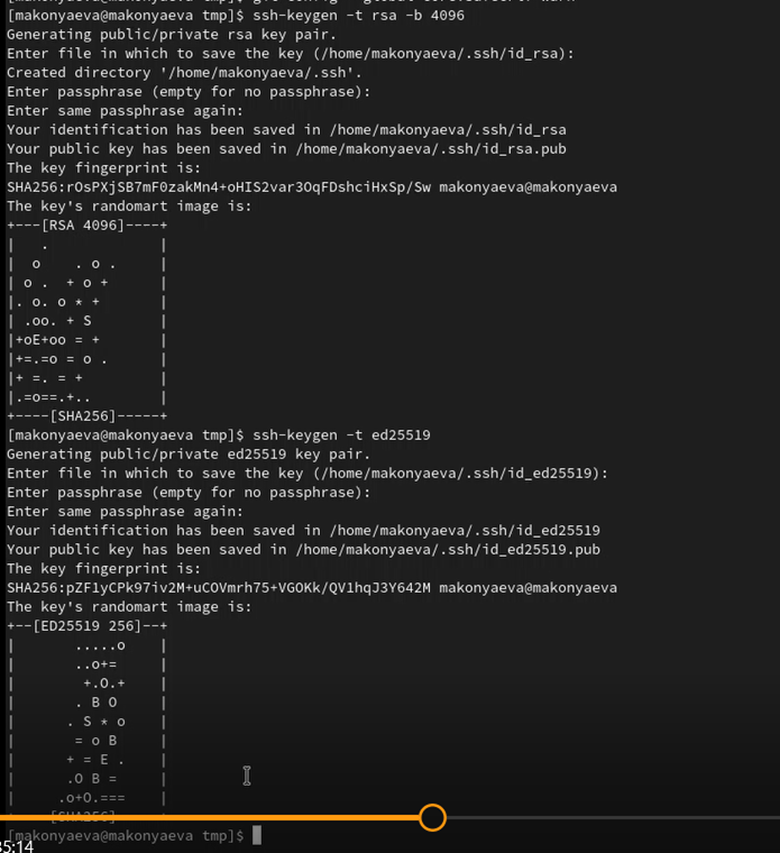
1. Установка пакета gh

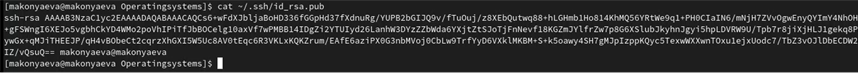
 *Изображение 4.1 Установка пакета gh*

1. Базовая настройка git, включающая имя, email, кодировку, имя начальной ветки, autocrlf и safecrlf

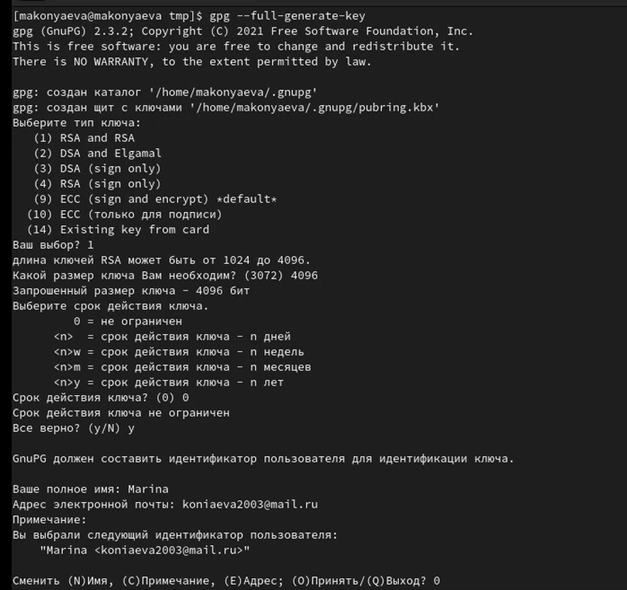
 *Изображение 5.1 Базовая настройка git*

1. Создание ssh ключей с двумя алгоритмами шифрования: RSA и ed25529

 *Изображение 6.1 Создание ssh ключей*

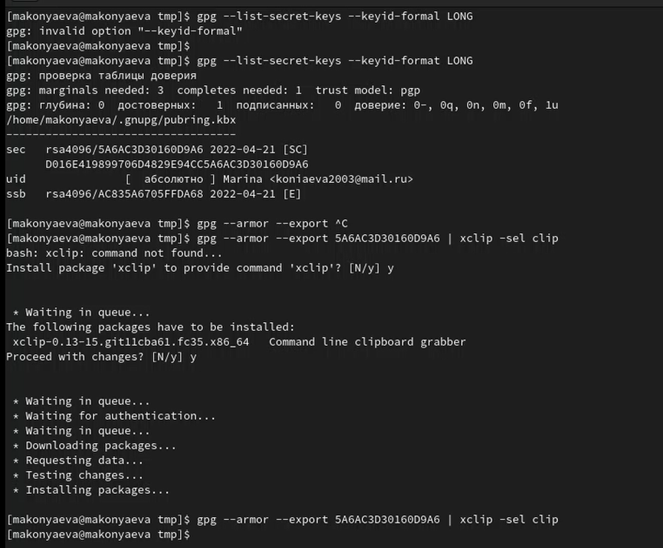
 *Изображение 6.2 Создание ssh ключей*

1. Создание pgp ключа



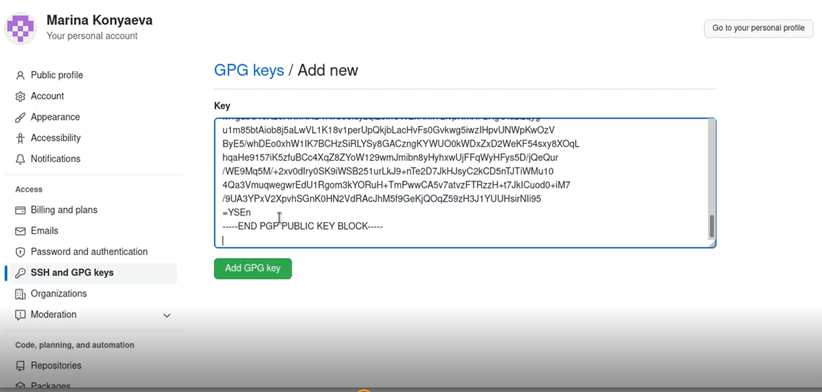
Создание pgp ключа

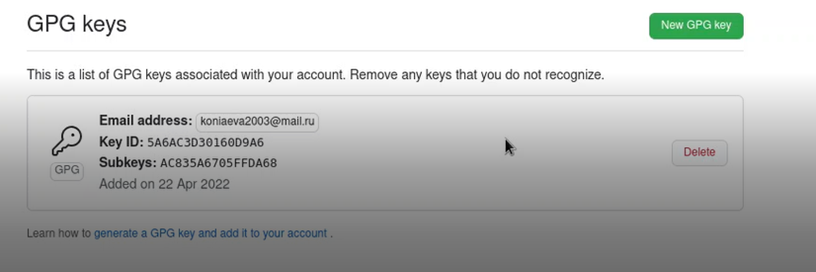
1. Добавление ключа PGP на Github



Добавление ключа PGP на Github

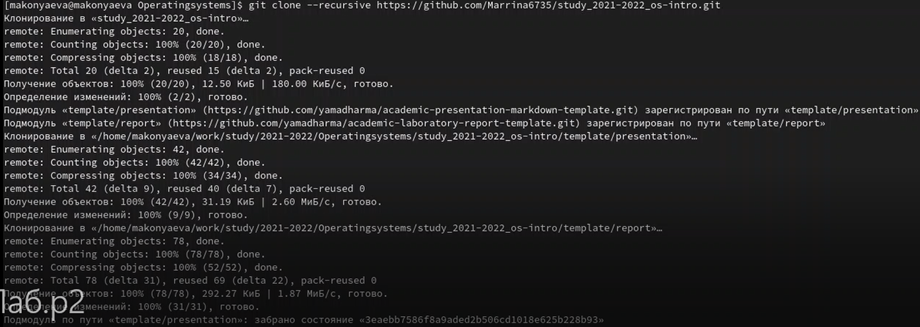
1. Просмотр отпечатка и копирования в буфер ключа pgp

 *Изображение 9.1 Просмотр отпечатка и копирования*

 *Изображение 9.2 Просмотр отпечатка и копирования*

1. Создание и клонирования репозитория

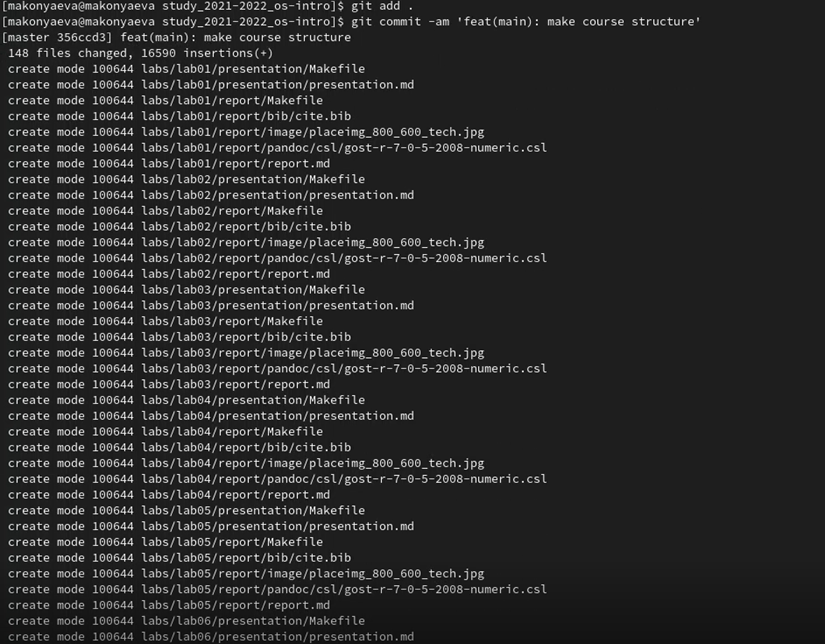
Создание и клонирования репозитория *Изображение 10.1 Создание и клонирования репозитория*

 *Изображение 10.2 Создание и клонирования репозитория*

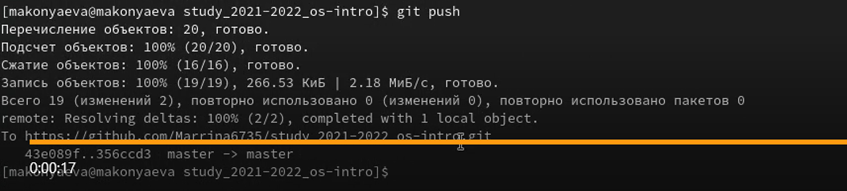
1. Настройки каталога курса, а именно, удаление json файла и создание структуры

Настройки каталога *Изображение 11.1 Настройки каталога*

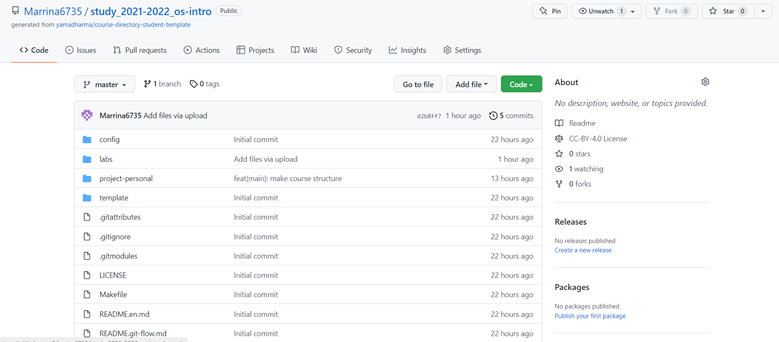
1. Создание commit

 *Изображение 12.1 Создание commit*

1. Push на сервер

 *Изображение 13.1 Push*

1. Проверим, что репозиторий появился на Github

 *Изображение 14.1 репозиторий на Github*

# Выводы

В ходе данной лабораторной работы я получила знания о системах контроля версий, изучила идеологию и применение данной технологии, также получила практические навыки по настройке системы и по созданию репозитория и обработки файлов.

# Контрольные вопросы

1. 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. 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 репозитарии, с дополнительной информацией.
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).
4. Изначально разработчик работает с веткой master. При реализации отдельных частей проекта может создать ветки для них. При завершении изменений разработчик коммитит и пушит изменения на сервер. Если разработка на сторонней ветке завершена, то её можно смерджить (merge), например с основной веткой master.
5. Каждый разработчик работает в своей ветке над отдельной частью проекта. Если на ветке работает несколько разработчиков они как правило достать изменения, сделанные другим и работать уже с ними (git pull). После завершения или заканчивая какой-то логический кусок они делают коммит и пушат на сервер. Если работа закончена необходимо смерджить ветки, например с главной веткой(master).
6. 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. – 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 изменения: 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 – сл ияние 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. Работа с таким удалённым репозиторием подразумевает выполнение стандартных операций отправки и получения, как и с любым другим удалённым репозиторием. Просмотр удалённых репозиториев. Для того, чтобы просмотреть список настроенных удалённых репозиториев, вы можете запустить команду git remote add . Когда вы клонируете репозиторий, команда clone автоматически добавляет этот удалённый репозиторий под именем «origin». Таким образом, git fetch origin извлекает все наработки, отправленные на этот сервер после того, как вы его клонировали (или получили изменения с помощью fetch).
9. ‘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.
10. Игнорировать файлы коммита можно с помощью файла .gitignore. Туда обычно помещаются файлы, которые занимают много места или не особо нужны для проекта. Например, картинки, папки с кешем, веса для нейросетей и другие файлы.