**РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ**

**Факультет физико-математических и естественных наук**

**ОТЧЕТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1**

*дисциплина: Операционные системы*

Студент: Сироджиддинов Камолиддин

Группа: НКНбд 01-21

МОСКВА 2022 г.

**Цель работы**

Изучить идеологию и применение средств контроля версий

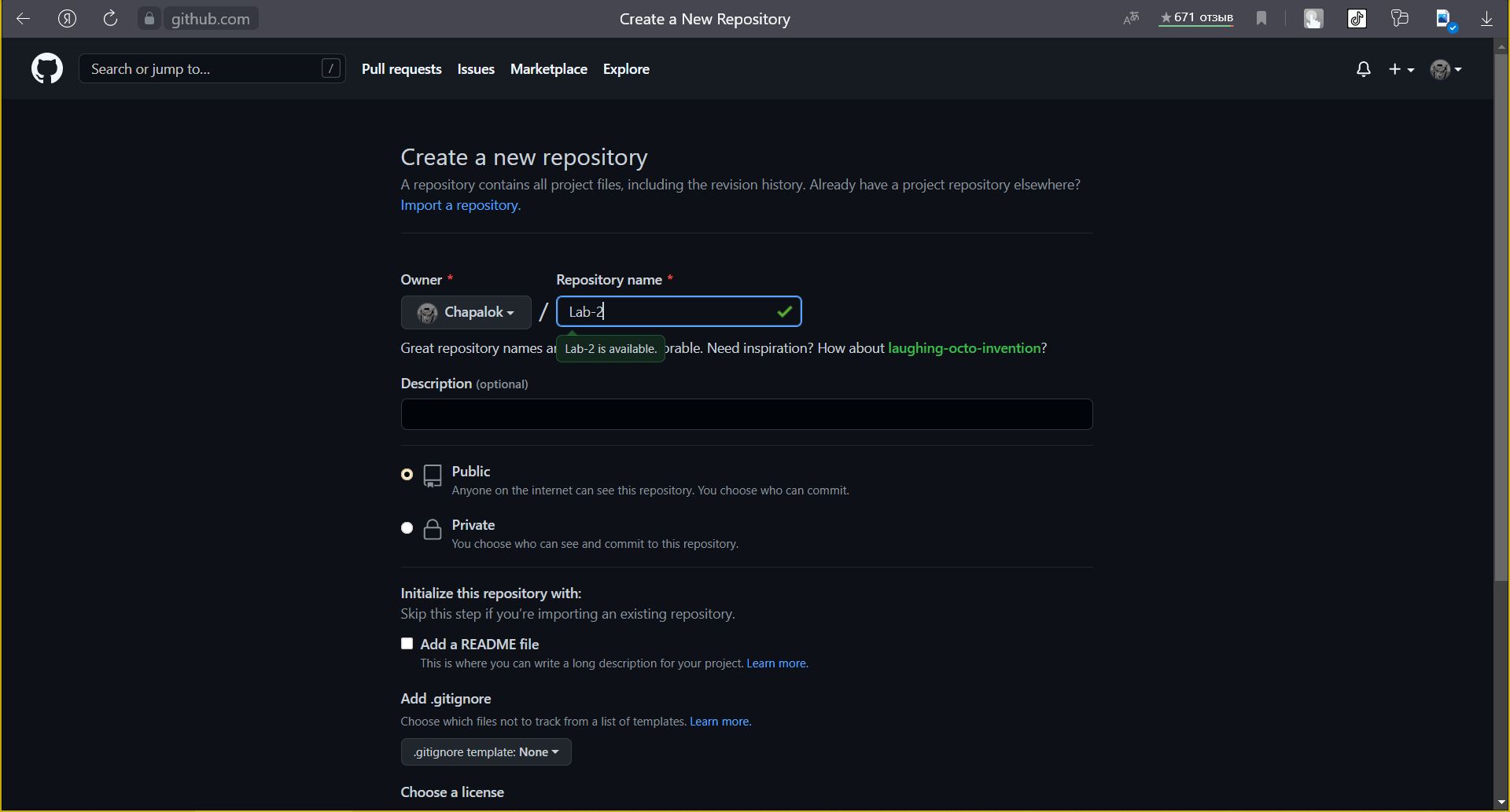
**Ход работы:**

Я создал учотную запись на сайте GitHub.

Получил ключ SSH:



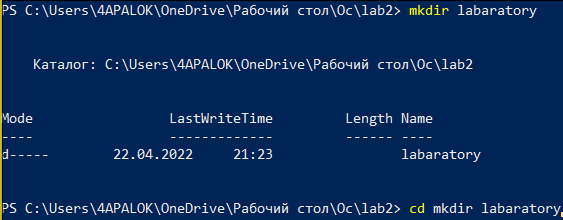
Создал репозиторий на сайте GitHub:



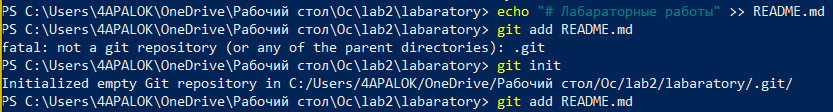
Сделал предварительную конфигурацию указав имя и email владельца репозитория:



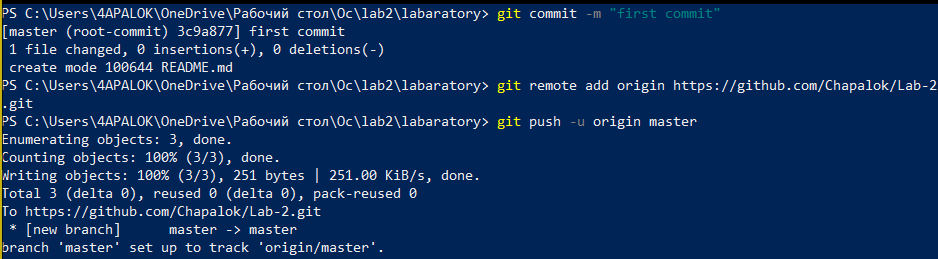
Cоздал рабочий каталог laboratory с помощью команды mkdir т перешел в него с помощью команды cd. Инициализировал системы



Создал заготовку для файла README.md



Сделал первый коммит и выложил его на GitHub.



Затем скачал шаблон для C++.



Добавил новый файл с помощью команды git add.



Выполнил коммит git commit -a



Отправил на Github.

Проверил что я на ветку master.



Записал версию с помощью команды echo “1.0.0” >> VERSION

Отправил на Github всё:



**Вывод:**

Я изучил идеологию и применение средств контроля версий

**Контрольные вопросы**

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

Система контроля версий (Version Control System, VCS) представляет собой программное обеспечение, которое позволяет отслеживать изменения в документах, при необходимости производить их откат, определять, кто и когда внес исправления и т.п.

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

Хранилище – это содержимое скрытой папки .git, где хранятся все версии рабочей области и служебная информация. Commit – это команда, с помощью которой можно сохранить текущее состояние проекта. Новая версия проекта сохраняется и добавляется в хранилище. Репозиторий хранит все рабочие файлы и историю их изменений. Директория .git/ с её содержимым относится к Git, а все остальные файлы называются рабочей копией и принадлежат пользователю.

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

Централизованные системы контроля версий предполагают сохранение версий проектов на общий сервер, с которого потом получают нужные версии клиенты. В децентрализованных системах контроля версий при каждом копировании удалённого репозитория (расположенного на сервере) происходит полное копирование данных в локальный репозиторий (установленный на рабочем компьютере). Каждая копия содержит все данные, хранящиеся в удалённом репозитории. В случае, возникновения технической неисправности на стороне сервера, удаленный репозиторий можно перезаписать с любой сохраненной копии. Примеры централизованных VCS: CVS, Subversion. Примеры децентрализованных VCS: Git, Bazaar, Mercurial.

1. Опишите порядок работы с общим хранилищем VCS.

Работа пользователя со своей веткой начинается с проверки и получения изменений из центрального репозитория (при этом в локальное дерево до начала этой процедуры не должно было вноситься изменений). Затем можно вносить изменения в локальном дереве и/или ветке. После завершения внесения какого-то изменения в файлы и/или каталоги проекта необходимо разместить их в центральном репозитории. Для этого необходимо проверить, какие файлы изменились к текущему моменту. при необходимости удаляем лишние файлы, которые не хотим отправлять в центральный репозиторий.

Затем полезно просмотреть текст изменений на предмет соответствия правилам ведения чистых коммитов. Если какие-либо файлы не должны попасть в коммит, то помечаем только те файлы,

изменения которых нужно сохранить. Затем сохраняем изменения, поясняя, что было сделано и отправляем в центральный репозиторий.

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

Сначала сделаем предварительную конфигурацию, указав имя и email владельца репозитория и настроив utf-8 в выводе сообщений git. Для инициализации локального репозитория необходимо ввести в командной строке: git init. После это в исходном каталоге появится каталог .git, в котором будет храниться история изменений. Создадим тестовый текстовый файл и добавим его в локальный репозиторий. 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.

1. Каковы основные задачи, решаемые инструментальным средством git?

Вот некоторые из многочисленных возможностей данных систем:

* Возврат к любой версии кода из прошлого.
* Просмотр истории изменений.
* Совместная работа без боязни потерять данные или затереть чужую работу.

1. Назовите и дайте краткую характеристику командам 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 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

– 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

1. Что такое и зачем могут быть нужны ветви (branches)?Ветвь в Git — это просто легковесный подвижный указатель на один из этих коммитов. Имя ветки по умолчанию в Git — master. Когда вы вначале создаёте коммиты, вам даётся ветвь master, указывающая на последний сделанный коммит. При каждом новом коммите, указатель двигается вперёд автоматически.
2. Как и зачем можно игнорировать некоторые файлы при commit?

Игнорируемые файлы – это, как правило, специфичные для платформы файлы или автоматически созданные файлы из систем сборки. Некоторые общие примеры включают в себя:

* Файлы времени выполнения, такие как журнал, блокировка, кэш или временные файлы.
* Файлы с конфиденциальной информацией, такой как пароли или ключи API.
* Скомпилированный код, такой как .class или .o.
* Каталоги зависимостей, такие как /vendor или /node\_modules.
* Создавать папки, такие как /public, /out или /dist.
* Системные файлы, такие как .DS\_Store или Thumbs.db
* Конфигурационные файлы IDE или текстового редактора.

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