**РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ**

**Факультет физико-математических и естественных наук**

**Кафедра прикладной информатики и теории вероятностей**

**ОТЧЕТ**

**по лабораторной работе № 2**

*дисциплина: Операционные системы*

Студент: Леон Атупанья Хосе Фернандо

Группа: НПМбд-02-20

**МОСКВА**

2021\_

**Цель работы**

Изучить идеологию и применение средств контроля версий.

**Ход работы**

Создайте учётную запись на <https://github.com>. (рисунок 1)

Изображение выглядит как текст, снимок экрана, монитор, экран

Автоматически созданное описание

Рисунок 1

Настроим систему контроля версий git. Поэтому во первых мы должны установить git на системе. (рисунок 2)

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2

Создаем наш репозиторий на Github.com (рисунок 3)

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3

Прежде создать наш локальный репозиторий, мы делаем предварительную конфигурацию, указав имя и email владельца репоитория, а на консоль пишем команд git congif –global user.name “leonjose” и git config –global user.email “[fernandoleon833@gmail.com](mailto:fernandoleon833@gmail.com)” (рисунок 4)

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4

После этого мы пишем код чтобы получить Key для последующей идентификации пользователя на сервере репозиториев, потом пишем код cat /home/Fernando/.ssh/id\_rsa.pub чтобы показать key. (рисунок 5).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5

Опять зайдем на Github.com и мы ввели код, который мы получили в разделе SSH and GPG Keys. Мы назначаем имя и код. (рисунок 6)

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 6

Создаем каталоги по имени program/2020-2021/OS/laboratory и используем коды echo “# lab-2” >> README.md, git init, git add, git add README.md. (рисунок 7)

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 7

После этого выкладываем на Gibhub. (рисунок 8)

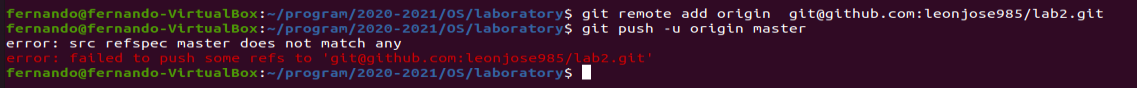


Рисунок 8

Добавим файл лицензии, шаблон игнорируемых файлов, новые файлы и выполним коммит и отправим на github. (рисунок 9,10)

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 9

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 10

А теперь начинаем с конфигурацией git flow и инициализируем, проверяем если мы на ветке develop, запишем версию, добавим индекс, зальём реализую ветку в основную ветку и отправим данные на git.hub. (рисунок 11 и 12)

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 11

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 12

**Выводы**

В ходе данной работы я научился о системе контроль версий и как можем работать с сервером Github.com, из того же образом я узнал, как создать локальный репозиторий и как связать его с сервером.

**Контрольные вопросы**

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

*Система контроля версий (Version Control System, VCS) — программное обеспечение для облегчения работы с изменяющейся информацией. VCS 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.*

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

*Система контроля версий (от англ. Version Control System, VCS) — это место хранения кода. Как dropbox, только для разработчиков!*

*Она заточена именно на разработку продуктов. То есть на хранение кода, синхронизацию работы нескольких человек, создание релизов (бранчей)... Но давайте я лучше расскажу на примере, чем она лучше дропбокса. Всё как всегда, история с кучей картиночек для наглядности. А потом я подробнее расскажу, как VCS работает — что значит "создать репозиторий", "закоммитить и смерджить изменения", и другие страшные слова. В конце мы пощупаем одну из систем VCS руками, скачаем код из открытого репозитория.*

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

*Централизованные системы контроля версий*

*Централизованные системы контроля версий представляют собой приложения типа клиент-сервер, когда репозиторий проекта существует в единственном экземпляре и хранится на сервере. Доступ к нему осуществлялся через специальное клиентское приложение. В качестве примеров таких программных продуктов можно привести CVS, Subversion.*

*Распределенные системы контроля версий*

*Распределенные системы контроля версий (Distributed Version Control System, DVCS) позволяют хранить репозиторий (его копию) у каждого разработчика, работающего с данной системой. При этом можно выделить центральный репозиторий (условно), в который будут отправляться изменения из локальных и, с ним же эти локальные репозитории будут синхронизироваться. При работе с такой системой, пользователи периодически синхронизируют свои локальные репозитории с центральным и работают непосредственно со своей локальной копией. После внесения достаточного количества изменений в локальную копию они (изменения) отправляются на сервер. При этом сервер, чаще всего, выбирается условно, т.к. в большинстве DVCS нет такого понятия как “выделенный сервер с центральным репозиторием”.*

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

*Следующая серьёзная проблема, с которой сталкиваются люди, это необходимость взаимодействовать с другими разработчиками. Для её решения были придуманы централизованные системы контроля версий. Примером такой системы является, например, хранилище конфигурации, которое использует платформа 1С:Предприятия 8, а также системы Visual SourceSafe, CVS, Subversion и Perforce. Они имеют один сервер, содержащий все версии файлов, и некоторое количество клиентов, которые извлекают файлы из этого центрального хранилища. Применение централизованных систем контроля версий являлось стандартом на протяжении многих лет.*

1. Опишите порядок работы с общим хранилищем VCS.

*Такие системы наиболее широко используются при разработке программного обеспечения для хранения исходных кодов разрабатываемой программы. Однако они могут с успехом применяться и в других областях, в которых ведётся работа с большим количеством непрерывно изменяющихся электронных документов. В частности, системы управления версиями применяются в САПР, обычно в составе систем управления данными об изделии (PDM). Управление версиями используется в инструментах конфигурационного управления (Software Configuration Management Tools).*

1. Каковы основные задачи, решаемые инструментальным средством git?

*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.

*Init: данная команда создаёт новый репозиторий.*

*Пример использования: init имя проекта.*

*Clone. Производит копирование существующего репозитория с удалённого источника. Вот так используется команда git clone: clone git://github.com/путь к репозиторию.*

*Add. Наиболее часто используемая команда в системе контроля версий Git. Она выполняет простую задачу — добавляет указанные файлы в особую область, именуемую индексом или сценой. В неё можно перенести несколько файлов или папок, которые необходимо будет впоследствии добавить в репозиторий или, выражаясь на языке Git,.*

1. Приведите примеры использования при работе с локальным и удалённым репозиториями.

*Удалённые репозитории представляют собой версии вашего проекта, сохранённые в интернете или ещё где-то в сети. У вас может быть несколько удалённых репозиториев, каждый из которых может быть доступен для чтения или для чтения-записи. Взаимодействие с другими пользователями предполагает управление удалёнными репозиториями, а также отправку и получение данных из них. Управление репозиториями включает в себя как умение добавлять новые, так и умение удалять устаревшие репозитории, а также умение управлять различными удалёнными ветками, объявлять их отслеживаемыми или нет и так далее. В данном разделе мы рассмотрим некоторые из этих навыков.*

1. Что такое и зачем могут быть нужны ветви (branches)?

*Ветви функциональностей (feature branches), также называемые иногда тематическими ветвями (topic branches), используются для разработки новых функций, которые должны появиться в текущем или будущем релизах. При начале работы над функциональностью (фичей) может быть ещё неизвестно, в какой именно релиз она будет добавлена. Смысл существования ветви функциональности (feature branch) состоит в том, что она живёт так долго, сколько продолжается разработка данной функциональности (фичи). Когда работа в ветви завершена, последняя вливается обратно в главную ветвь разработки (что означает, что функциональность будет добавлена в грядущий релиз) или же удаляется (в случае неудачного эксперимента)*.

1. 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.*