Архитектура компьютеров и операционные системы | Операционные системы

Первоначальна настройка git

Акрур Имад НКАбд-01-24

Содержание

Список иллюстраций

Список таблиц

# 1 Цель работы:

* Научиться оформлять отчёты с помощью легковесного языка разметки **Markdown**.

# 2 Задания:

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

# 3 Теоретическое введение:

## 3.1 Markdown:

* **Markdown** — язык разметки текстов. Такие тексты легко писать и читать. Их можно без труда сконвертировать в HTML. Большинство программистов предпочитают Markdown для написания документации, описаний своих проектов, написания блогов и так далее.

## 3.2 Зачем это нужно?

1. Для добавления разметки туда, где невозможна реальная разметка. Например, в простом текстовом файле или в тех же СМС, где невозможно выделение жирным, создание заголовков, выделение цитат и пр.
2. Для более удобного написания текстов для последующей конвертации в HTML или другие форматы.

# 4 Выполнение лабораторной работы:

## 4.1 Цель второй лабораторной работы:

* Изучить идеологию и применение средств контроля версий.
* Освоить умения по работе с git.

## 4.2 Теоретическое введение:

### 4.2.1 Примеры использования git:

* Система контроля версий **Git** представляет собой набор программ командной строки. Доступ к ним можно получить из терминала посредством ввода **команды git** с различными опциями.
* Благодаря тому, что **Git** является распределённой системой контроля версий, резервную копию локального хранилища можно сделать простым копированием или архивацией.

### 4.2.2 Выполнение второй лабораторной работы:

#### 4.2.2.1 Установка программного обеспечения :

##### 4.2.2.1.1 Установка git:

* На этом шаге мы должны были установить **git** через консоль (в нашем случае **git** уже был установлен) (рис. fig. 1)

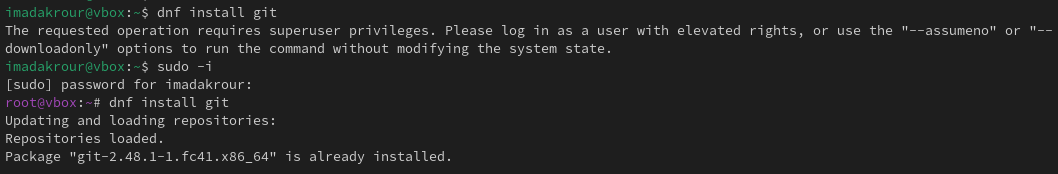


Рис. 1: рисунок 1

##### 4.2.2.1.2 Установка gh:

* Затем нам пришлось скачать **gh** (рис. fig. 2)

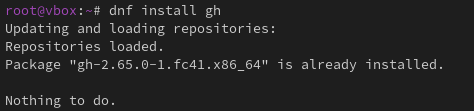


Рис. 2: рисунок 2

#### 4.2.2.2 Базовая настройка git:

* на этом шаге я ничего не делал, потому что мой репозиторий уже был настроен. (рис. fig. 3)

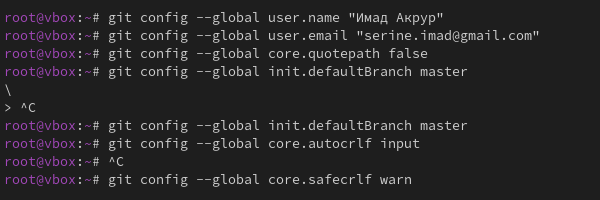


Рис. 3: рисунок 3

#### 4.2.2.3 Создать ключи ssh:

* Для **ssh-ключа** было то же самое, поэтому мне пришлось пропустить этот шаг

#### 4.2.2.4 Создать gpg ключ:

* На этом шаге мы должны были сгенерировать \*\* gpg ключ \*\*, введя следующие команды (рис. fig. 4)

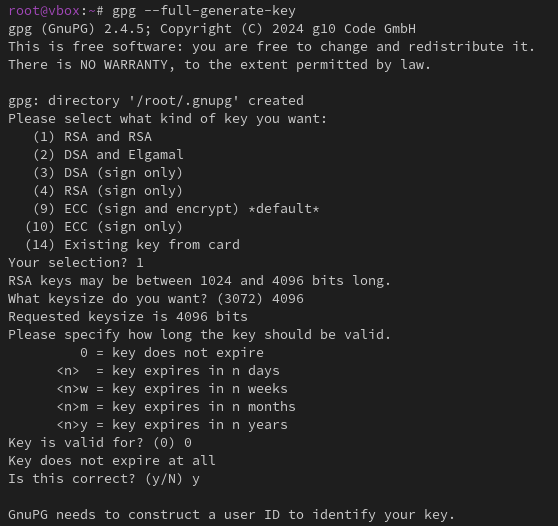


Рис. 4: рисунок 4

* затем нам пришлось ввести кодовую фразу для защиты нашего ключа (рис. fig. 5)

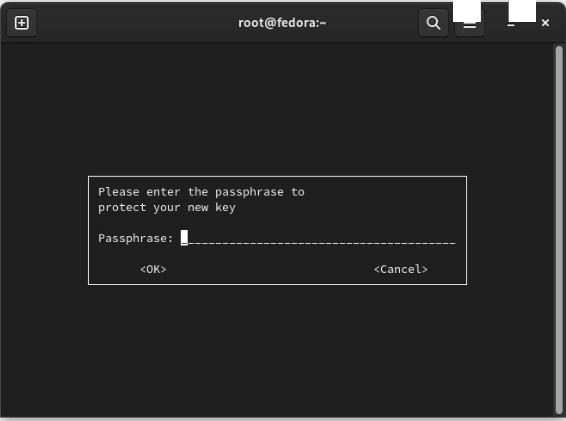


Рис. 5: рисунок 5

#### 4.2.2.5 Настройка github:

* Я пропустил этот шаг, потому что я уже создал **учетную запись github** раньше.
* все мои данные были заполнены в учетной записи.(рис. fig. 6)

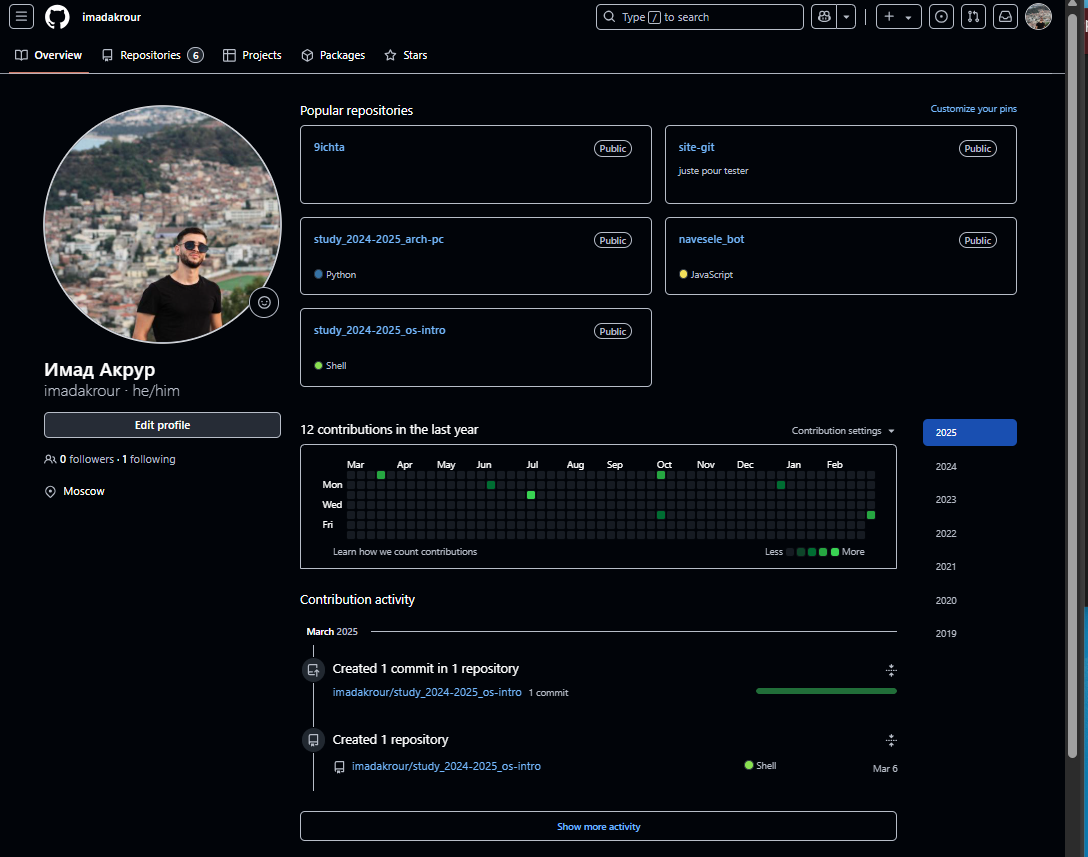


Рис. 6: рисунок 6

#### 4.2.2.6 Добавление PGP ключа в GitHub:

* На этом шаге мы хотели добавить **ключ pgp** в нашу учетную запись github, поэтому нам пришлось скопировать отпечаток ключа (рис. fig. 7)

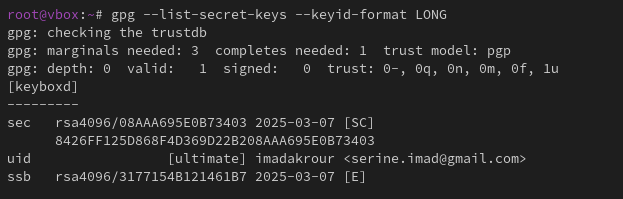


Рис. 7: рисунок 7

* после этого мы открыли **github** и в **настройках** добавили наш ключ **PGP** (рис. fig. 8)

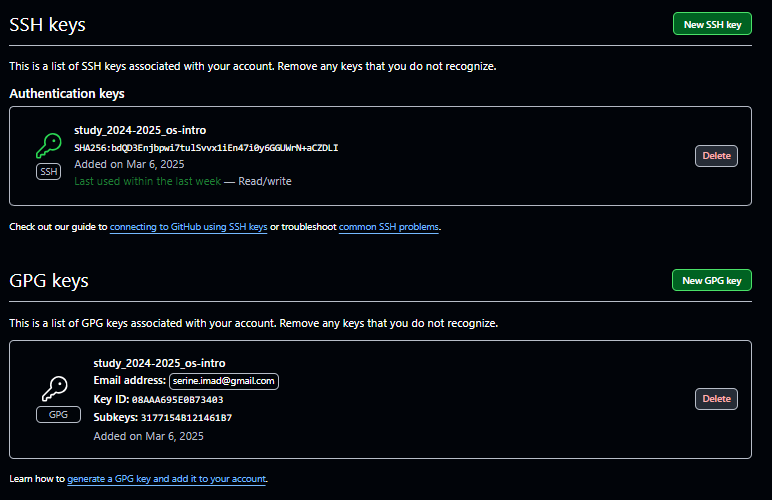


Рис. 8: рисунок 8

#### 4.2.2.7 Настройка автоматических подписей коммитов git и gh:

* На этом шаге мы хотели заставить **github** использовать **наш введёный email**, в качестве подписи при отправке коммитов.
* для этого нам пришлось создать токен аутентификации, чтобы иметь доступ к нашей учетной записи git через консоль (рис. fig. 9) (рис. fig. 10)

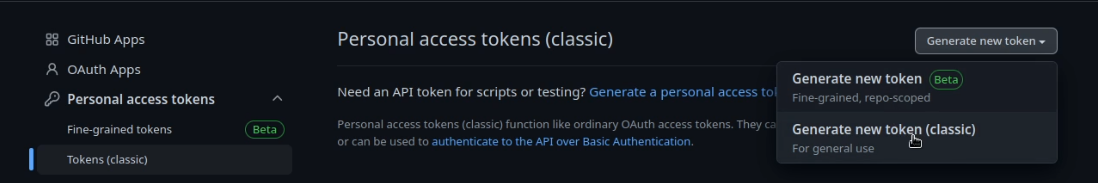


Рис. 9: рисунок 9

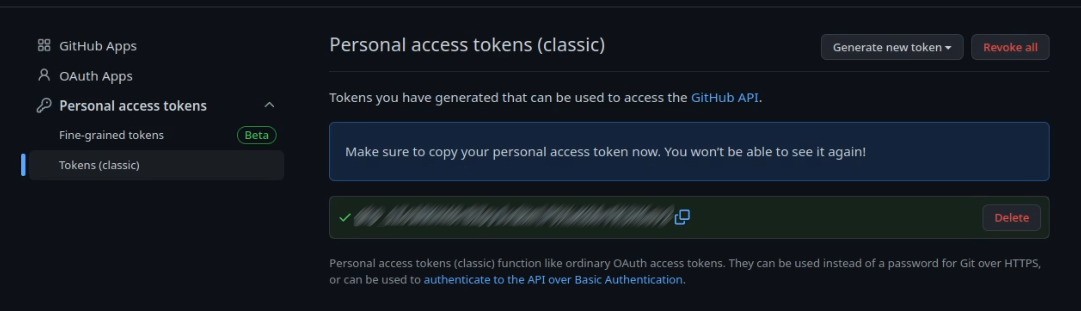


Рис. 10: рисунок 10

* и после этого мы вставили токен в консоль, которая дала нам доступ к нашей учетной записи github (рис. fig. 11)

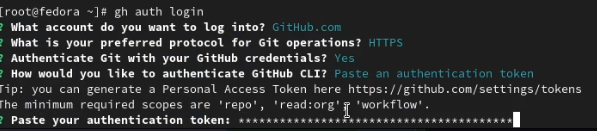


Рис. 11: рисунок 11

#### 4.2.2.8 Сознание репозитория курса на основе шаблона:

* Для меня этот путь *~/work/study/2022-2023/“Операционные системы”* уже был создан, поэтому я сразу ввел вторую команду.
* На этом шаге мне пришлось пройти аутентификацию с помощью **токена**, чтобы получить **доступ**, затем я клонировал глобальный репозиторий в свой локальный (рис. fig. 12) (рис. fig. 13)

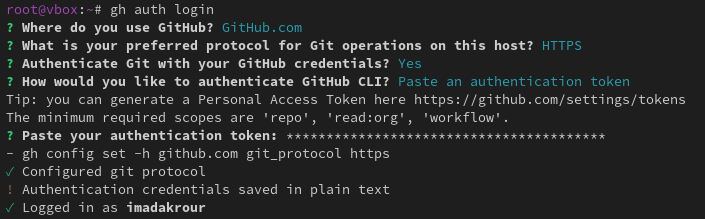


Рис. 12: рисунок 12

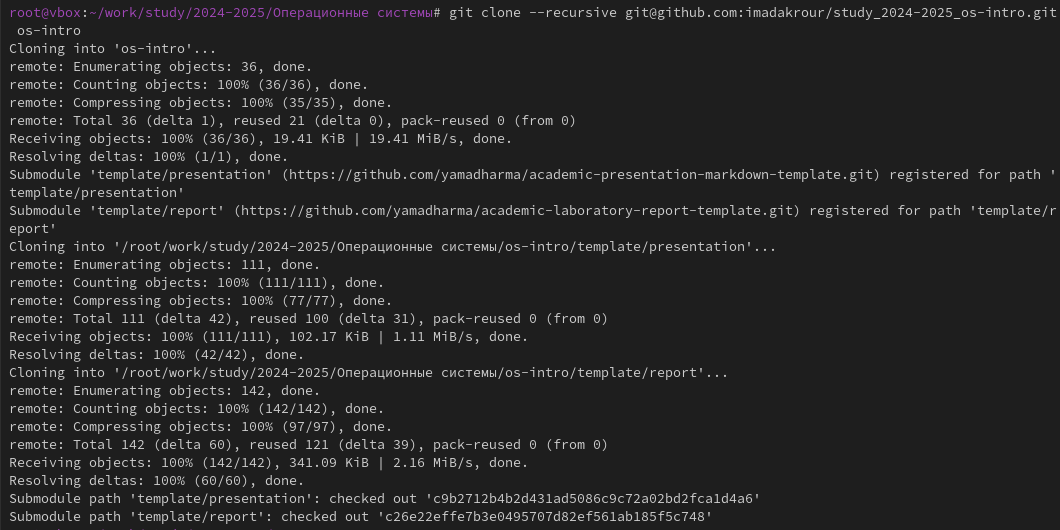


Рис. 13: рисунок 13

#### 4.2.2.9 Настройка каталога курса:

* На этом шаге я попытался удалить файл package.json, но он уже был удален, поэтому я немедленно отправил другие изменения (рис. fig. 14)

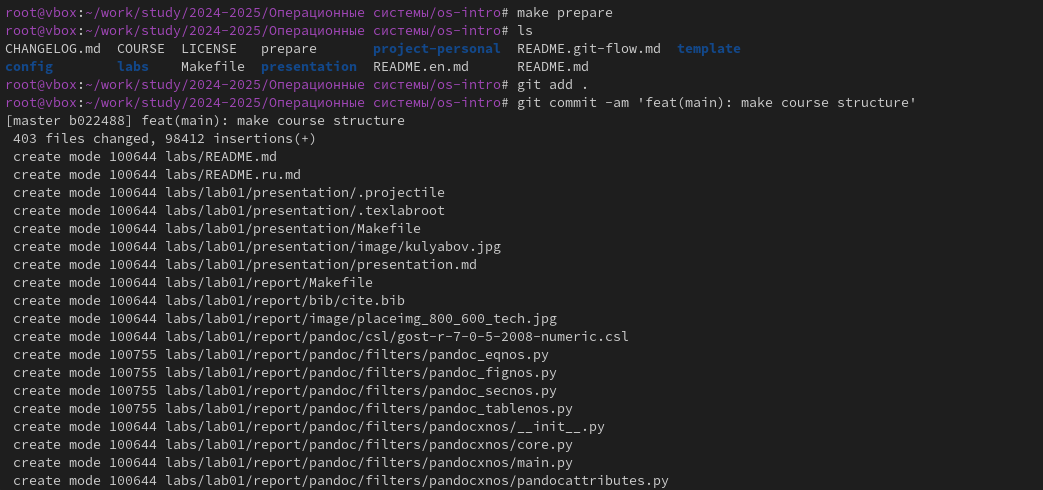


Рис. 14: рисунок 14

## 4.3 Контрольные вопросы:

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

* Системы контроля версий (Version Control System, VCS) применяются при работе нескольких человек над одним проектом. Обычно основное дерево проекта хранится в локальном или удалённом репозитории, к которому настроен доступ для участников проекта. При внесении изменений в содержание проекта система контроля версий позволяет их фиксировать, совмещать изменения, произведённые разными участниками проекта, производить откат к любой более ранней версии проекта, если это требуется.

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

* ранилище (repository, сокр. repo), или репозитарий, — место хранения всех версий и служебной информации.
* commit : создание новой версии («сделать коммит», «закоммитить»)
* история журнал : перечень версий можно вернуться к любой.
* Рабочая копия (working copy или working tree) — текущее состояние файлов проекта, основанное на версии из хранилища (обычно на последней).

1. Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида.(рис. fig. 15)

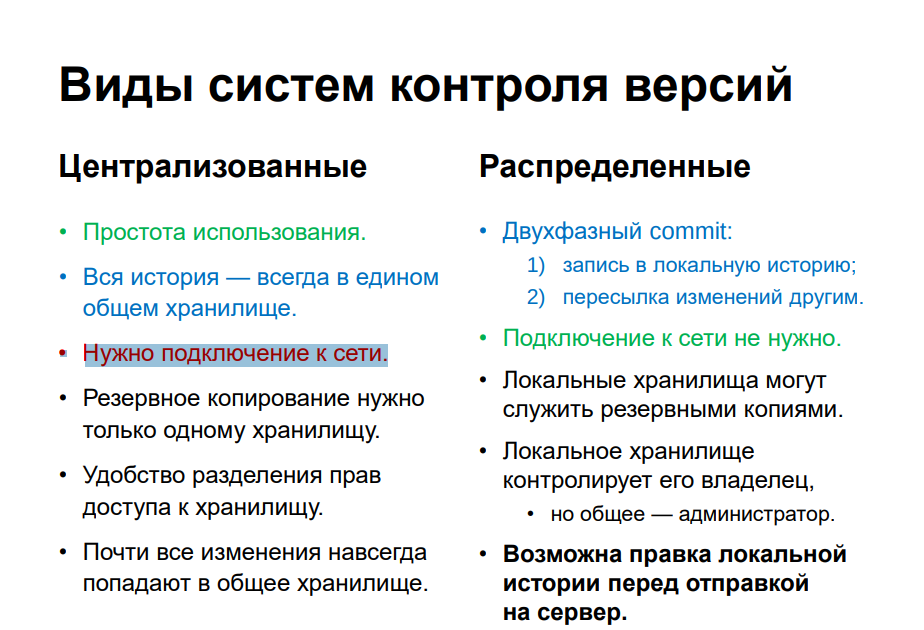


Рис. 15: рисунок 15

1. Опишите действия с VCS при единоличной работе с хранилищем.(рис. fig. 16)

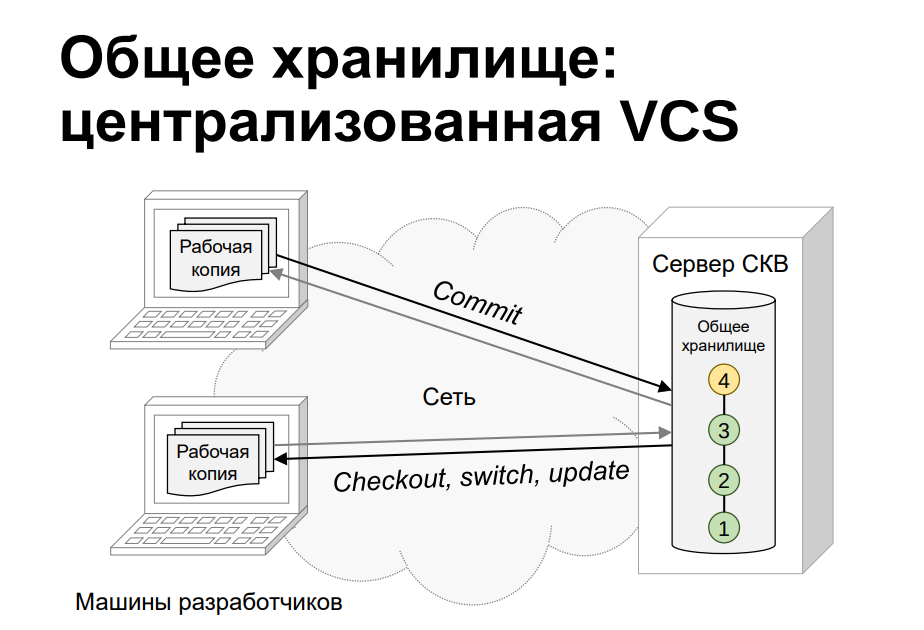


Рис. 16: рисунок 16

1. Опишите порядок работы с общим хранилищем VCS.(рис. fig. 17)

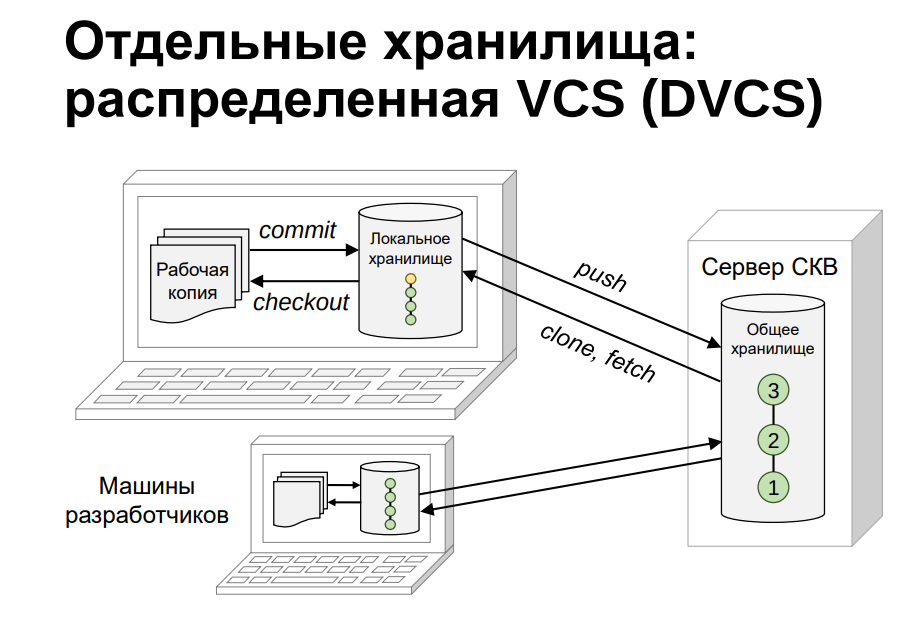


Рис. 17: рисунок 17

1. Назовите и дайте краткую характеристику командам git.

* Перечислим 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 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

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

* Ветвь (branch) — это линейный участок истории.
* Ветвь по умолчанию называется master.О том, зачем нужны другие ветви и как история может быть нелинейной, см. далее.
* В выводе git log отмечен конец ветви.

1. Как и зачем можно игнорировать некоторые файлы при commit?

* Существуют различные типы файлов, которые мы, возможно, захотим, чтобы git проигнорировал перед фиксацией, например, файлы, связанные с нашими пользовательскими настройками или любыми настройками утилиты, личные файлы, такие как пароли и ключи API. Эти файлы никому больше не нужны, и мы не хотим загромождать наш git. Мы можем сделать это с помощью “**.gitignore**”

## 4.4 выводы по результатам выполнения заданий:

* после выполнения этих упражнений мы смогли применить на практике наши знания, которые мы получили о git и системе контроля версий в целом.

# 5 Выводы, согласованные с целью работы:

* к концу лабораторной работы этой лабораторной работе мы узнали, как использовать markdown для создания pdf-файлов быстрее и эффективнее.