**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: Гнатюк Анастасия Станиславовна**

**Группа: НФИбд-01-21**

**МОСКВА**

2022 г.

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

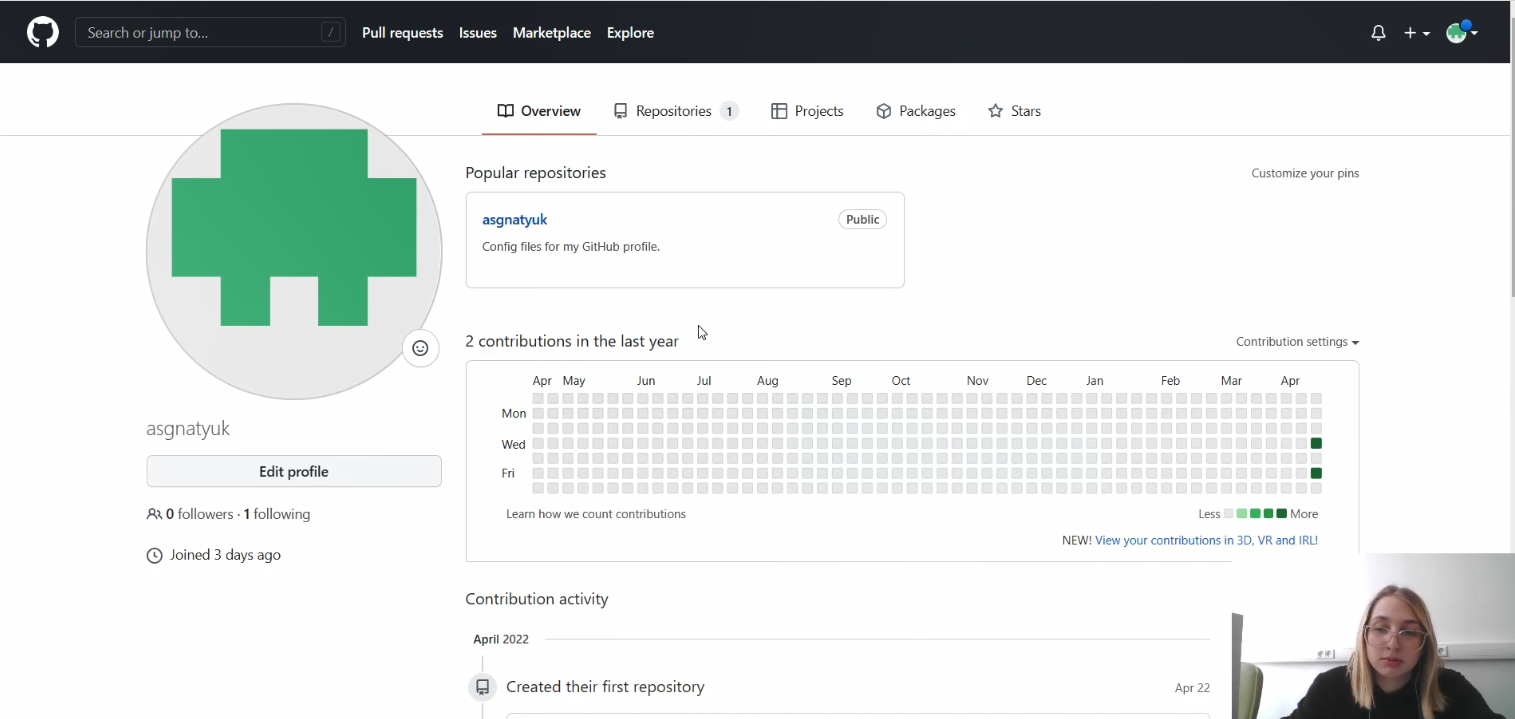
**Последовательность выполнения работы:**

1. Создать базовую конфигурацию для работы с git.

**Настройка github**

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

2. Заполните основные данные на <https://github.com>.

Создаём учётную запись на GitHub и заполняем основные данные. (Рис.1.1)

**Рис.1.1**: Учётная запись на GitHub

**Установка программного обеспечения.**

Установка git-flow в Fedora Linux

– Это программное обеспечение удалено из репозитория.

– Необходимо устанавливать его вручную:

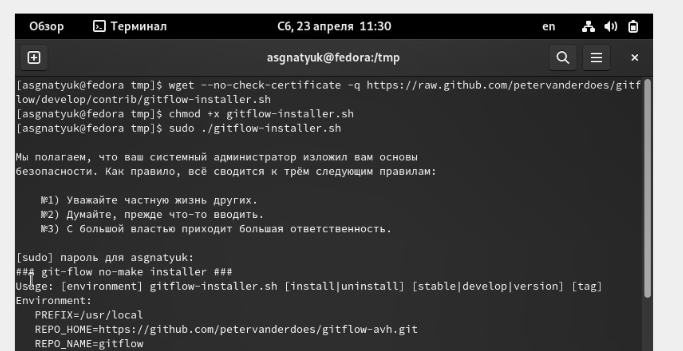
1 cd /tmp

2 wget --no-check-certificate -q https://raw.github.com/petervanderdoes ⌋ ↪ /gitflow/develop/contrib/gitflow-installer.sh

3 chmod +x gitflow-installer.sh

4 sudo ./gitflow-installer.sh install stable

Вводим данные в верху команды и устанавливаем git-flow в Fedora Linux(Рис.1.2)



**Рис.1.2:** Установка git-flow

**Установка gh в Fedora Linux**

1 sudo dnf install gh

Базовая настройка git

– Зададим имя и email владельца репозитория:

1 git config --global user.name "Name Surname"

2 git config --global user.email "work@mail"

– Настроим utf-8 в выводе сообщений git:

1 git config --global core.quotepath false – Настройте верификацию и подписание коммитов git.

– Зададим имя начальной ветки (будем называть её master):

git config --global init.defaultBranch master

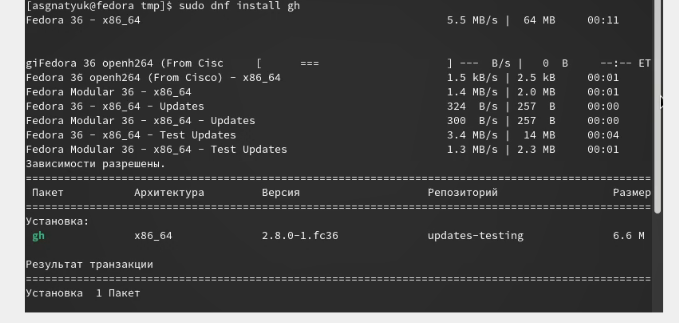
– Параметр autocrlf:

1 git config --global core.autocrlf input

– Параметр safecrlf:

1 git config --global core.safecrlf warn

Устанавливаем gh в Fedora Linux с помощью команды sudo dnf install gh(Рис.1.3)

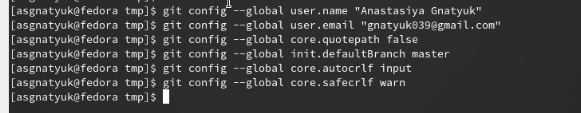


**Рис.1.3**: Установка gh

Далее делаем базовую настройку git. Задаём имя и email владельца репозитория(я):

1 git config --global user.name "Name Surname"

2 git config --global user.email "work@mail" (Рис.1.4)



**Рис.1.4**

Настраиваем utf-8 в выводе сообщений git:

1 git config --global core.quotepath false

– Настройте верификацию и подписание коммитов git.

– Задаём имя начальной ветки (будем называть её master):

git config --global init.defaultBranch master

– Параметр autocrlf:

1 git config --global core.autocrlf input

– Параметр safecrlf:

1 git config --global core.safecrlf warn(всё на Рис.1.4)

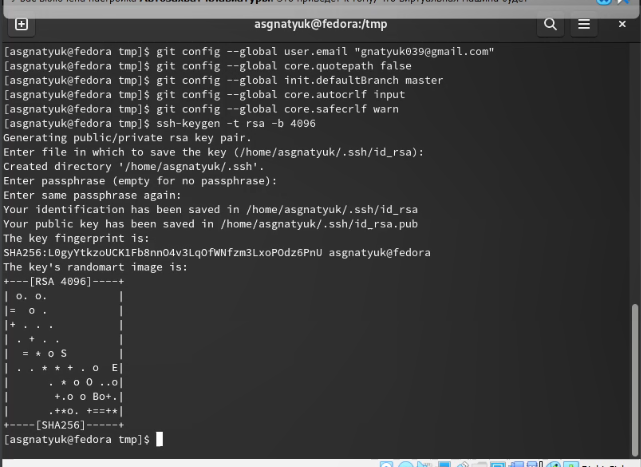
1. Создать ключи ssh

– по алгоритму rsa с ключём размером 4096 бит:

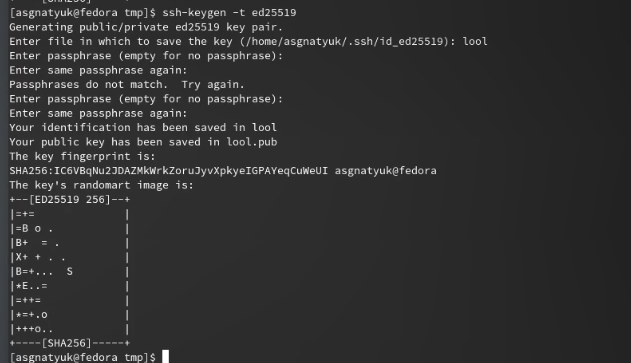
1 ssh-keygen -t rsa -b 4096

– по алгоритму ed25519:

1 ssh-keygen -t ed25519

Создание ключа ssh по алгоритму rsa –b 4096(Рис.2.1)

**Рис.2.1**

А затем по второму алгоритму(Рис.2.2)

**Рис.2.2**

1. Создать ключи pgp

**Генерируем ключ**:

gpg --full-generate-key

**Из предложенных опций выбираем:**

– тип RSA and RSA;

– размер 4096;

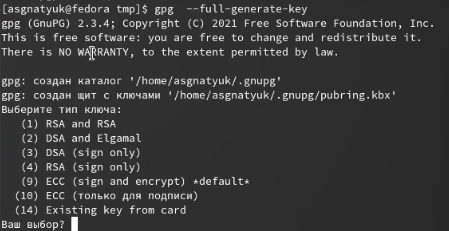
– выберите срок действия; значение по умолчанию— 0 (срок действия не истекает никогда).

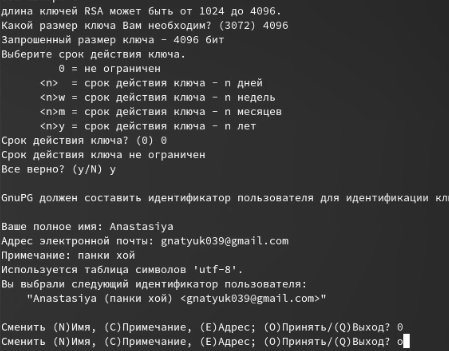
– GPG запросит личную информацию, которая сохранится в ключе:

– Имя (не менее 5 символов).

– Адрес электронной почты.

– При вводе email убедитесь, что он соответствует адресу, используемому на GitHub.

– Комментарий. Можно ввести что угодно или нажать клавишу ввода, чтобы оставить это поле пустым.

**Рис.3.1**

**Рис.3.2**

Делаем всё так, как нас просят в задании. Все опции выбираем правильно. (Рис.3.2)

**Добавление PGP ключа в GitHub**

– Выводим список ключей и копируем отпечаток приватного ключа:

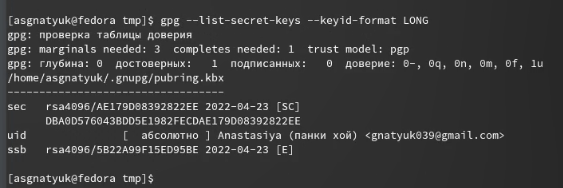
1 gpg --list-secret-keys --keyid-format LONG (Рис.3.3)

– Отпечаток ключа — это последовательность байтов, используемая для идентификации более длинного, по сравнению с самим отпечатком ключа.

– Cкопируйте ваш сгенерированный PGP ключ в буфер обмена:

 - мой отпечаток ключа

1 gpg --armor --export <PGP Fingerprint> | xclip -sel clip(Рис.3.4)

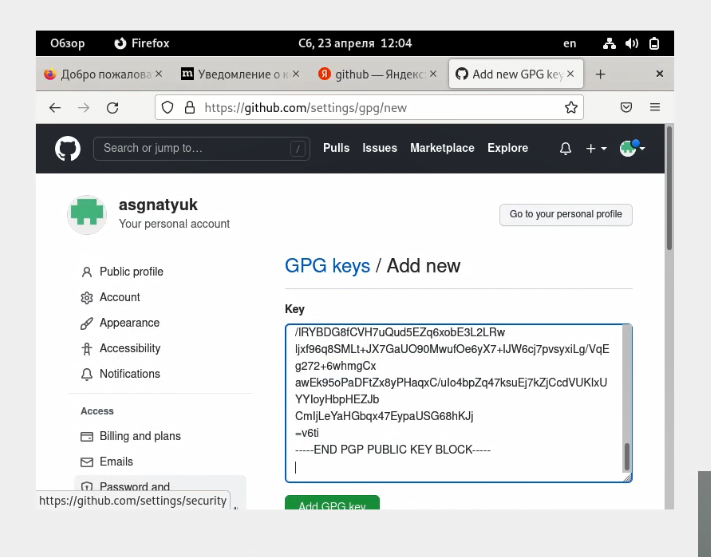
– Перейдите в настройки GitHub (https://github.com/settings/keys), нажмите на кнопку New GPG key и вставьте полученный ключ в поле ввода. (Рис.3.5)

**Рис.3.3**: Вывод списка ключей



**Рис.3.4**: Копируем сгенерированный код в буфер обмена

Переходим в настройки GitHub (https://github.com/settings/keys), нажимаем на кнопку New GPG key и вставляем полученный ключ в поле ввода.



**Рис.3.5**

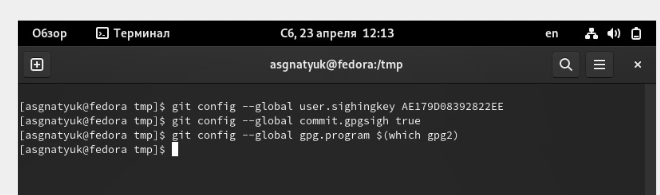
**4)**Настроить подписи git

– Используя введёный email, укажите Git применять его при подписи коммитов:

1 git config --global user.signingkey <PGP Fingerprint>

2 git config --global commit.gpgsign true

3 git config --global gpg.program $(which gpg2)(Рис.4.1)



**Рис.4.1**

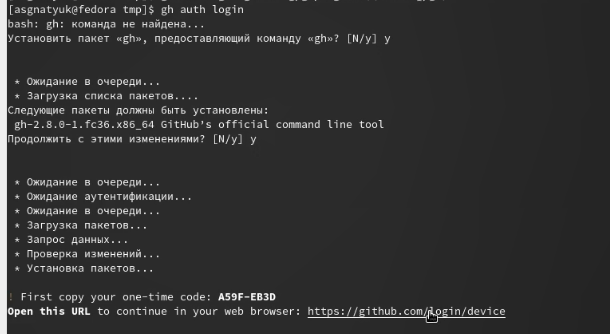
**Настройка gh**

– Для начала необходимо авторизоваться

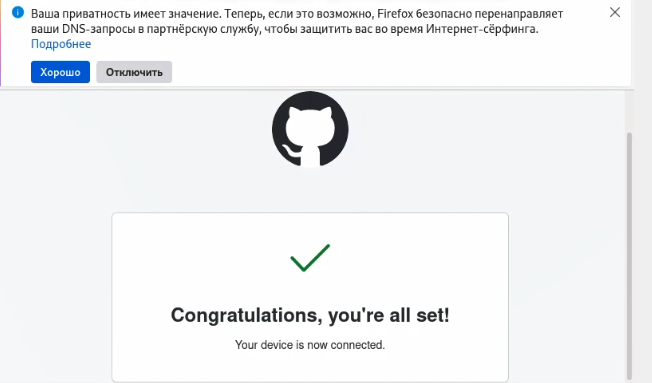
1 gh auth login(Рис.4.2)

– Утилита задаст несколько наводящих вопросов.

– Авторизоваться можно через браузер. (Рис.4.3)



**Рис.4.2**



**Рис.4.3**

**Сознание репозитория курса на основе шаблона**

**–** Необходимо создать шаблон рабочего пространства.

– Например, для 2021–2022 учебного года и предмета «Операционные системы» (код предмета os-intro) создание репозитория примет следующий вид:

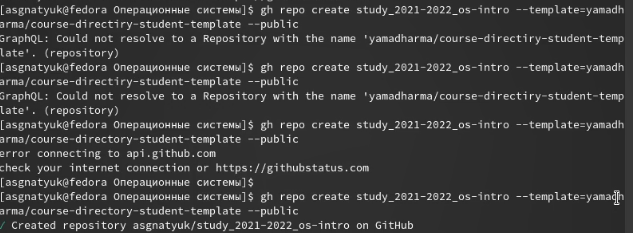
1 mkdir -p ~/work/study/2021-2022/"Операционные системы"

2 cd ~/work/study/2021-2022/"Операционные системы"

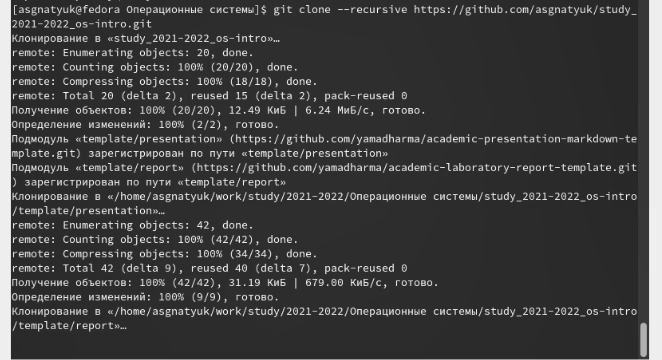
3 gh repo create study\_2021-2022\_os-intro

↪ --template=yamadharma/course-directory-student-template --public

4 git clone --recursive

↪ git@github.com:<owner>/study\_2021-2022\_os-intro.git os-intro

**Рис.4.4**



**Рис.4.5**

**5)** Настройка каталога курса

**– Перейдите в каталог курса**:

1 cd ~/work/study/2021-2022/"Операционные системы"/os-intro (Рис.5.1)

**– Удалите лишние файлы**:

1 rm package.json (Рис.5.2)

**– Создайте необходимые каталоги:**

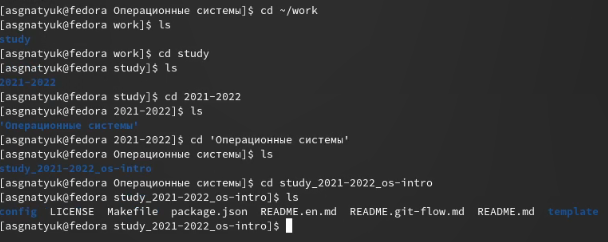
1 make COURSE=os-intro (Рис.5.3)

**– Отправьте файлы на сервер**: (Рис.5.4 – 5.5)

1 git add .

2 git commit -am 'feat(main): make course structure'

3 git push



**Рис.5.1**



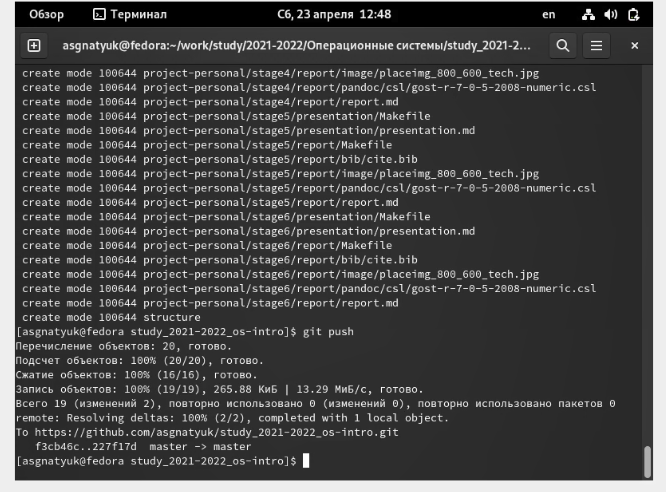
**Рис.5.2**



**Рис.5.3**



**Рис.5.4**



**Рис.5.5**

**Вывод:** Я изучила идеологию и применение средств контроля версий и освоила умения по работе с git.

**Контрольные вопросы:**

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

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 репозитарии, 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.

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. Опишите порядок работы с общим хранилищем VCS.

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. 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 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

9. 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.

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

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.

**Контрольные вопросы**