\*Git:\*

1. Перечисли команды для следующих действий:

- создание ветки

- удаление ветки

- создание коммита

- индексация изменений

- слияние ветки

- подключение внешнего репозитория

- отправка данных в ВР

- получение данных из ВР

Ответ:

- создание ветки =git branch имя\_ветки

- удаление ветки = git branch -d имя\_ветки

- создание коммита = git add -A либо git commit -m "Commit message."

- индексация изменений = git add и git rm

- слияние ветки = git merge имя\_ветки

- подключение внешнего репозитория = git remote add origin имя\_репозитория

- отправка данных в ВР = git push

- получение данных из ВР = git fetch

2.Чем гит отличается от других SVC?

Ответ:

Основное отличие Git от любой другой СКВ (включая Subversion и её собратьев) — это подход к работе со своими данными. Концептуально, большинство других систем хранят информацию в виде списка изменений в файлах. Эти системы (CVS, Subversion, Perforce, Bazaar и т. д.) представляют хранимую информацию в виде набора файлов и изменений, сделанных в каждом файле, по времени (обычно это называют контролем версий, основанным на различиях).

3. Опиши разницу между мерджем и ребейзом

Ответ:

Rebase нужен для порядка (линейность истории), чтоб потом смотреть и понимать. rebase переносит коммиты поверх того, что было в результирующей ветке. merge оставляет коммиты, как есть, просто добавляя связующий коммит между ветками.

4. Опиши основные стратегии ветвления.

Ответ:

GitHub Flow стратегия

По GitHub Flow получается, что прежде чем начать работу над чем-то новым, будь то исправление или новая фича, нужно создать новую ветку на основе master’а и дать ей подходящее имя.

Далее, начинается работа над реализацией. Нужно постоянно отправлять коммиты на удаленный сервер с тем же именем. Когда приходит понимание, что все готово, нужно создать пул-реквест в master ветку. Потом хотя бы один, а лучше — два человека должны посмотреть этот код и нажать Approve. Обычно обязательно должен посмотреть тимлид проекта и кто-то еще, и тогда уже можно завершать пул-реквест.

GitFlow стратегия

Предыдущая стратегия (GitHub Flow) была по сути не очень сложной. Есть два типа веток: master и фиче ветки. А вот GitFlow уже серьёзнее. Как минимум из картинки выше вы это можете понять) Итак, как работает эта стратегия? В целом GitFlow состоит из двух постоянных веток и нескольких типов временных веток (В контексте GitHub Flow, master ветка — постоянная, а другие — временные).

\*Docker: \*

1. Чем контейнеризация отличается от виртуализации?

Ответ:

Контейнеры и виртуальные машины — очень похожие между собой технологии виртуализации ресурсов. Виртуализация — это процесс, при котором единичный системный ресурс, такой как оперативная память, ЦП, диск или сеть, может быть «виртуализирован» и представлен в виде множества ресурсов. Основное отличие контейнеров и виртуальных машин заключается в том, что виртуальные машины виртуализируют весь компьютер вплоть до аппаратных уровней, а контейнеры — только программные уровни выше уровня операционной системы.

2. Напиши команды для:

- сборки образа

- старта контейнера

- остановки контейнера

- удаления контейнера

- удаления образа

- получения образа из репозитория

- отправки образа в репозиторий

- запуска команды внутри контейнера

Ответ:

- сборки образа = docker build -t

- старта контейнера = docker container start [OPTIONS] CONTAINER [CONTAINER...]

- остановки контейнера = docker stop [option] container\_id

- удаления контейнера = docker rm ID\_or\_Name ID\_or\_Name

- удаления образа = docker rmi

- получения образа из репозитория = [docker pull](https://docs.docker.com/engine/reference/commandline/pull/)

- отправки образа в репозиторий = sudo docker push

- запуска команды внутри контейнера = docker exec -it id\_контейнера bash\_команда

\*Dockerfile\*

1. Что такое - корневой или же родительский образ?

Ответ:

Родительский образ — это образ, на основе которого готовится наш образ. Нижний слой указывается командой FROM в Dockerfile.

2. Перечисли команды, которые создают новые слои

Ответ:

Новые слои в итоговом образе создаются только инструкциями FROM, RUN, COPY, ADD

3. Чем entrypoint отличается от cmd?

Ответ:

CMD -> это аргументы, добавленные к ENTRYPOINT, если пользователь не указывает какой-либо настраиваемый аргумент, например: docker run Ubuntu <custom\_cmd> в этом случае вместо добавления того, что указано в изображение в разделе CMD, докер запустит ENTRYPOINT <custom\_cmd>.

\*Docker-compose\*

1. Перечисли команды для:

- создания, старта и удаления контейнера

- выполнения команды внутри контейнера

- удаления контейнера и образа

Ответ:

- создания, старта и удаления контейнера :

создания, старта= docker compose up, [docker compose create](https://docs.docker.com/engine/reference/commandline/compose_create/)

удаления контейнера = docker compose rm

- выполнения команды внутри контейнера = [docker compose events](https://docs.docker.com/engine/reference/commandline/compose_events/) или [docker compose run](https://docs.docker.com/engine/reference/commandline/compose_run/)

- удаления контейнера и образа = docker compose down

2. Перечисли разделы docker-compose.yml

Ответ:

build, deploy, depends\_on, networks и тд

3. Напиши минимальный docker-compose.yml, который будет работать

Ответ:

# Версия Docker API

**version**: **'3.7'**

# Сервисы которые мы будем запускать

**services**:

# Первый сервис - db

**db**:

# Образ на основе которого он будет запускаться

**image**: postgres:12-alpine

# volumes - магическая вещь, которая создает некоторое устройство в

# рамках Docker и монтирует его в директорию /var/lib/postgresql/data

**volumes**:

- postgres\_data:/var/lib/postgresql/data/

# Переменные окружения

**environment**:

**POSTGRES\_USER**: movie

**POSTGRES\_PASSWORD**: **123456**

**POSTGRES\_DB**: movie

# Говорим открыть снаружи порт 5432

**expose**:

- **5432**

# Второй сервис - app

**app**:

# Говорим что его надо будет собрать - в качестве контекста

# передаем текущую директорию - в ней лежит Dockerfile

**build**: .

# Монтируем локальную директорию ./src в директорию

# внутри контейнера /opt/app

**volumes**:

- ./src:/opt/app

# Говорим пробросить порт 8000 хоста в порт 8000 контейнера

**ports**:

- **8000**:**8000**

# Зависит от сервиса db - запускать после него

**depends\_on**:

- db

# Просто говорим создать volume с именем postgres\_data

**volumes**:

**postgres\_data**:

\*Linux\*

1. Чем ОС Linux отличается от других ос?

Ответ:

Linux отличается от других ОС важной чертой – все опции с настройками расположены в одной точке доступа. Интуитивный, доброжелательный интерфейс помогает юзерам не тратить много времени на поиск той или иной опции по всему функционалу компьютера.

2. Что такое ядро?

Ответ:

Ядро — это своего рода главная программа, являющаяся основной частью операционной системы.

Оно выступает в роли посредника между устройствами компьютера (процессором, видеокартой, оперативной памятью и т.д.) и его программным обеспечением, абстрагируя от обычных программ и пользователей сложную, низкоуровневую работу с «железом» компьютера, предоставляя взамен простой, понятный и удобный в использовании интерфейс.

3. Что такое загрузчик?

Ответ:

Загрузчик - это небольшая программа, которая помогает загрузить операционную систему в память. Без загрузчика это сделать намного сложнее. В Linux существует несколько загрузчиков, которые мы можем использовать.

4. Опиши порядок загрузки Linux

Ответ:

#### 1. BIOS

* BIOS отвечает за базовый ввод/вывод данных с устройств/на устройства.
* Делает некоторые проверки целостности устройств. К тому же, за тестирование работоспособности электроники отвечает POST (Power-on self-test, он же «тест на адекватность себя самого», выполняющийся как этап пре-загрузки), который управляется BIOS
* Ищет, загружает и выполняет программу-загрузчик ОС
* Берет загрузчик из флопика, сидюка или жесткого диска. Во время загрузки BIOS'а вы можете нажать на кнопку (обычно это F12 или F2 или Del, зависит от платформы), если вам требуется внести некоторые изменения касательно настройки железа.
* Как только загрузчик был обнаружен и загружен в память, BIOS передает управление ему.
* Короче говоря, BIOS загружает и выполняет загрузочную запись (MBR).

#### 2. MBR

* MBR — это главная загрузочная запись, хранящаяся на жестком диске
* Она размещена в 1-м секторе загрузочного диска, например /dev/hda или /dev/sda
* MBR занимает меньше, чем 512 байтов. Она состоит из трех компонентов: 1) главная загрузочная информация, «живущая» в первых 446 байтах; 2) информация о таблице разделов — в следующих 64 байтах; 3) и последние 2 байта нужны для проверки корректности mbr.
* Она содержит информацию о GRUB'е (или LILO).
* Простыми словами — MBR загружает и выполняет загрузчик GRUB.

#### 3. GRUB

* GRUB — Grand Unified Bootloader.
* Если в вашей системе установлено более, чем одно ядро, у вас есть возможность выбирать, которое из них должен выполняться
* GRUB отображает ~~красивую анимацию plymouth~~ заставку, и, подождав несколько секунд интерактивного воздействия пользователя, если он не нажал ни одной клавиши, он загружает ядро, установленное по умолчанию в файле конфигурации grub.
* GRUB понимает, что такое файловая система (древние загрузчики Linux'а, например, LILO этого не понимают).
* Конфигурационный файл Grub обычно лежит по пути /boot/grub/grub.conf (так же /etc/grub.conf может быть символьной ссылкой на него). Вот пример файла конфигурации для CentOS:

#boot=/dev/sda<br>default=0<br>timeout=5<br>splashimage=(hd0,0)/boot/grub/splash.xpm.gz<br>hiddenmenu<br>title CentOS (2.6.18-194.el5PAE)<br>         root (hd0,0)<br>         kernel /boot/vmlinuz-2.6.18-194.el5PAE ro root=LABEL=/<br>         initrd /boot/initrd-2.6.18-194.el5PAE.img

* В качестве примечания к информации выше, конфигурационный файл содержит путь к ядру и образу initrd
* Если быть кратким, GRUB просто напросто загружает и выполняет образы ядра и initrd.

#### 4. Ядро или Kernel

* Ядро монтирует файловую систему в соответствии с настройкой «root=» в фале grub.conf
* Выполняет программу /sbin/init
* Поскольку init — это первый процесс, запущенный ядром Linux, поэтому она имеет идентификатор процесса (PID) №1. Можете выполнить «ps -ef | grep init» и убедиться в этом.
* initrd — это Initial RAM Disk, он же временный диск в оперативной памяти
* initrd используется самим ядром в качестве временной корневой файловой системы, пока kernel не загрузится в реальную примонтированную файловую систему. Этот временный диск также содержит необходимые для загрузки драйверы, позволяющие получить доступ к разделам дисков и другому оборудованию

#### 5. Init

* Смотрит в файл /etc/inittab для того, чтобы определить уровень выполнения (run level).
* Есть следующие уровни выполнения:  
  + 0 – прервать выполнение
  + 1 – Однопользовательский режим, так называемый «Single user mode», или иными словами, консоль восстановления
  + 2 – Многопользовательский режим без поддержки NFS
  + 3 – Полноценный многопользовательский режим
  + 4 – не используется
  + 5 – X11
  + 6 – перезагрузка
* Init определяет уровень выполнения по умолчанию исходя из /etc/inittab и использует его для загрузки всех необходимых программ.
* Выполните «grep initdefault /etc/inittab» на вашей системе, и вы узнаете, какой уровень по умолчанию у вас используется
* Если у вас не получается жить спокойно, то можете установить стандартный уровень на 0 или 6. :)
* В большинстве случаев вам будет достаточно уровня 3 или 5.

#### 6. Уровень выполнения программ (Runlevel)

* Когда Линукс выполняет свою загрузку, вы можете наблюдать загрузку различных служб. К примеру, это могут быть сообщения типа «starting Postfix … OK» (запускается Postfix). Эти службы — и называются программами уровня выполнения, выполняемые из директории, которая соответствует нужному уровню выполнения.
* Исходя из настроек по умолчанию, система будет выполнять файлы в соответствии с нижеприведенными директориями.  
  + Выполнение уровня 0 – /etc/rc.d/rc0.d/
  + Выполнение уровня 1 – /etc/rc.d/rc1.d/
  + Выполнение уровня 2 – /etc/rc.d/rc2.d/
  + Выполнение уровня 3 – /etc/rc.d/rc3.d/
  + Выполнение уровня 4 – /etc/rc.d/rc4.d/
  + Выполнение уровня 5 – /etc/rc.d/rc5.d/
  + Выполнение уровня 6 – /etc/rc.d/rc6.d/
* Но имейте ввиду, что еще в каталоге /etc могут быть символические ссылки. Например, /etc/rc0.d залинкован на /etc/rc.d/rc0.d.
* В каталогах /etc/rc.d/rc\*.d/ вы можете увидеть список программ, имя которых начинается из букв S и K.
* Программы, начинающиеся на S используются для запуска. S, потому что **s**tartup.
* Программы, которые начинаются с литеры K используются — правильно — для завершения работы. K, потому что **k**ill.
* Еще есть номера рядом с буквами S и K в именах программ. Эти номера используются для определения порядка запуска этих программ.
* К примеру, S12syslog предназначен для запуска демона syslog, его порядковый номер 12. S80sendmail — для запуска демона sendmail, имеющего порядковый номер 80. Таким образом, программа syslog будет запущена перед sendmail.

5. Опиши устройство ядра

Ответ:

Архитектура ядра Linux. На сегодняшний день Linux — монолитное ядро с поддержкой загружаемых модулей. Драйверы устройств и расширения ядра обычно запускаются в 0-м кольце защиты, с полным доступом к оборудованию. Все драйверы и подсистемы работают в своем адресном пространстве, отделенном от пользовательского.

6. Опиши порядок действий, в котором ты будешь искать логи неизвестной тебе программы в файловой системе

Ответ:

Сначала посмотрю активные логи ls -l /var/log/

Далее я воспользуюсь командой less а дальше напишу лог неизвестной мне программы