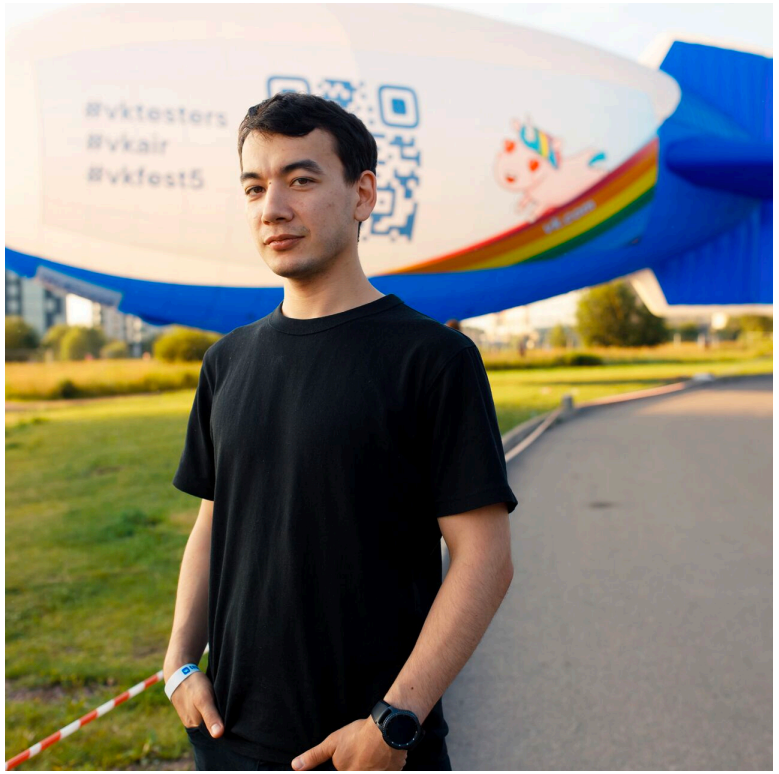
Как приручить Android emulator

Иван Левиков

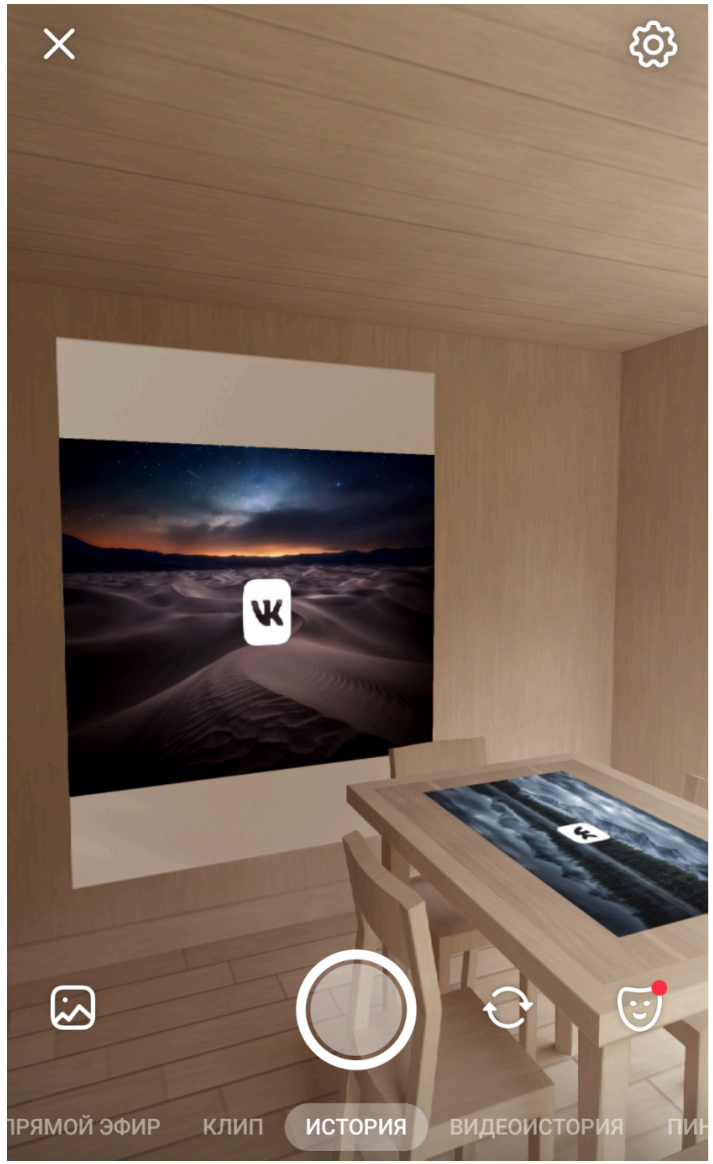


Развиваю и ускоряю автотесты,
прокачиваю инфраструктуру

vk.com/li
li@vk.com

О чем поговорим

1. Для чего нам нужны эмулятор
2. Собираем наш эмулятор
3. Вносим изменения в эмулятор и сокращаем вес docker образа
4. Полезные ссылки



Эмулятор, как часто мы используем их

Как часто вы задумывались, что значат параметры в конфигурационных файлах?

Дисклеймер

Кому будет полезен мой доклад?

Тем, кто сталкивался или готовятся столкнуться с эмуляторами на базе ОС Android

Что нужно помнить при внедрении?

- При сборке нового базового образа его нужно настраивать по аналогии с предыдущими, потому что при первоначальной сборке нет миграции
- При сборке образа **желательно использовать** версионирование, ведь при необходимости можно вернуться к предыдущей жизнеспособной версии
- Эмуляторы собираются под X86, но благодаря Google на них можно запустить ARM, x86

ПОГНАЛИ

Собираем наш эмулятор

Что нам понадобится

Docker	Терминал и ноутбук с Linux	KVM (QEMU)	Среда для работы
docs.docker.com/engine/install/		Для виртуализации	jetbrains.com/fleet/

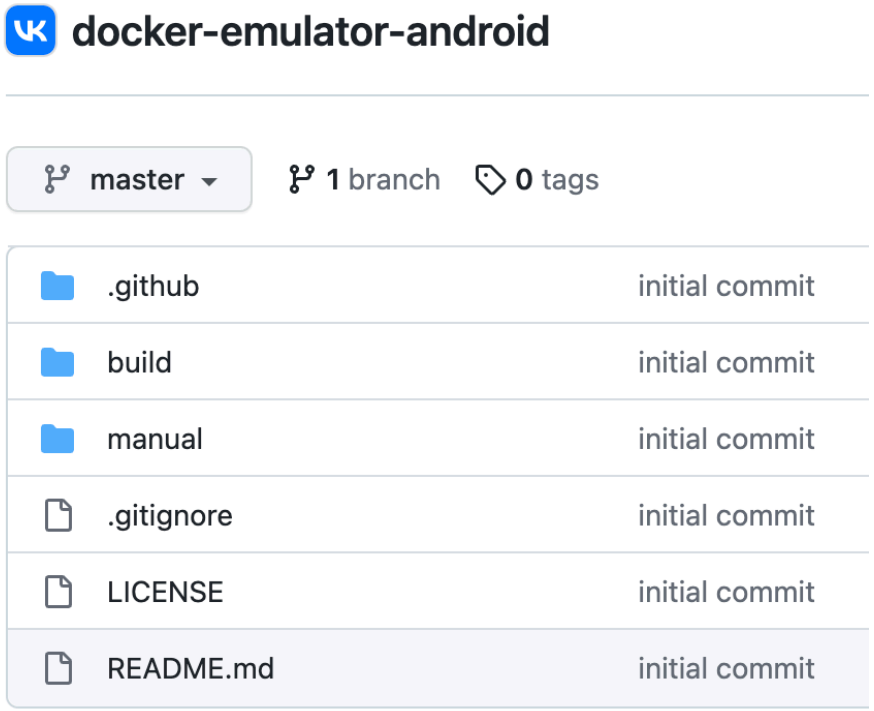
Наше окружение

Мы используем Kubernetes для оркестрации

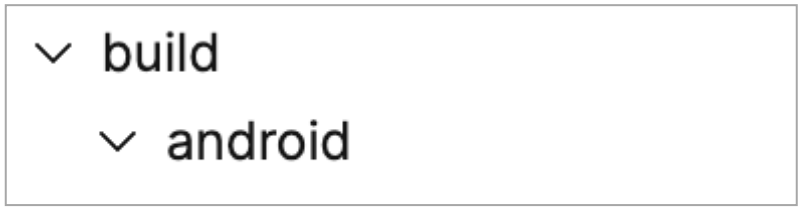
- Ноды = Сервера
- Поды ~ Эмулятор

Git clone

git clone https://github.com/VKCOM/docker-emulator-android.git

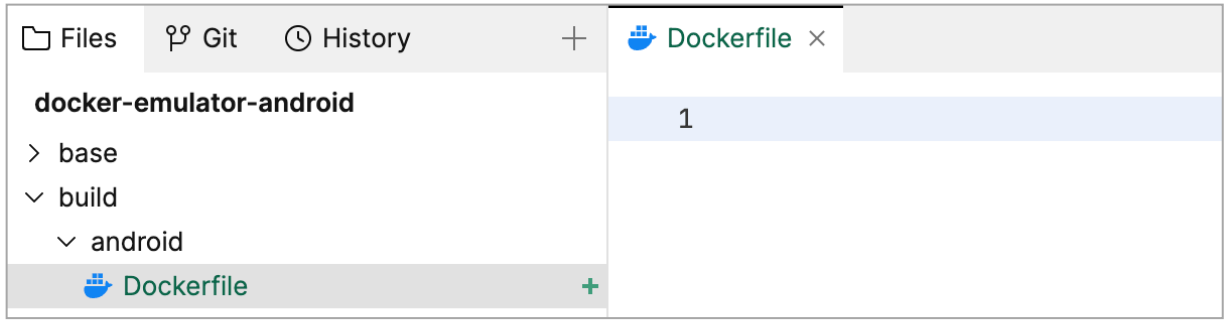


Создаем рабочую директорию



Для начала, создадим новую папку android

Создаем докер файл

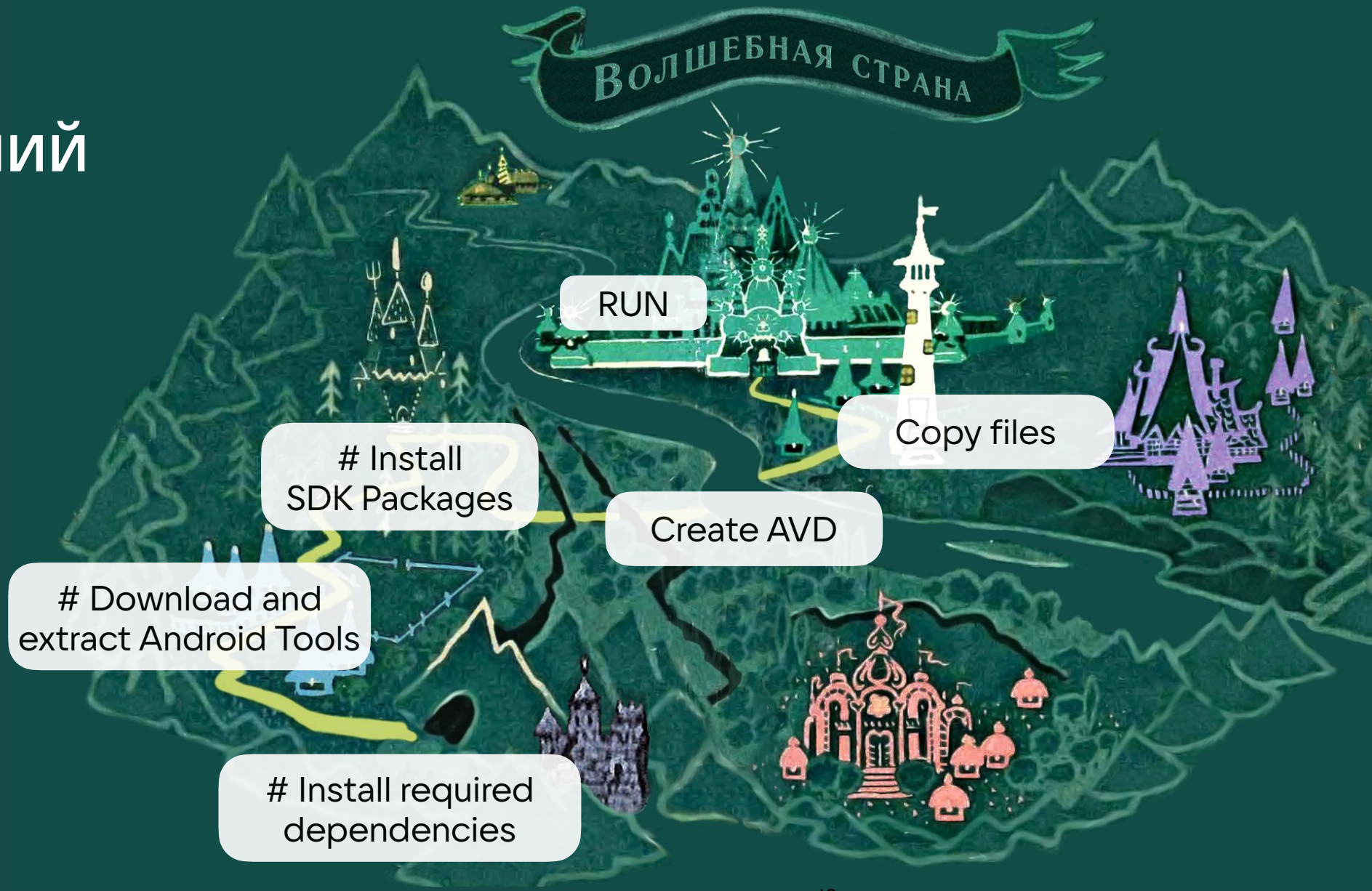


После создадим новый файл в директории, где будут храниться у вас все файлы, необходимые для эмуляторов.

Мы будем использовать легкую конфигурацию, в которой я вынес в adb команды большинство рутинных операций.

Если вы хотите что-то убрать или добавить меняйте значения в нужных местах, обозначенных в документации.

Масштаб приключений



Выбираем базовый образ ОС Linux

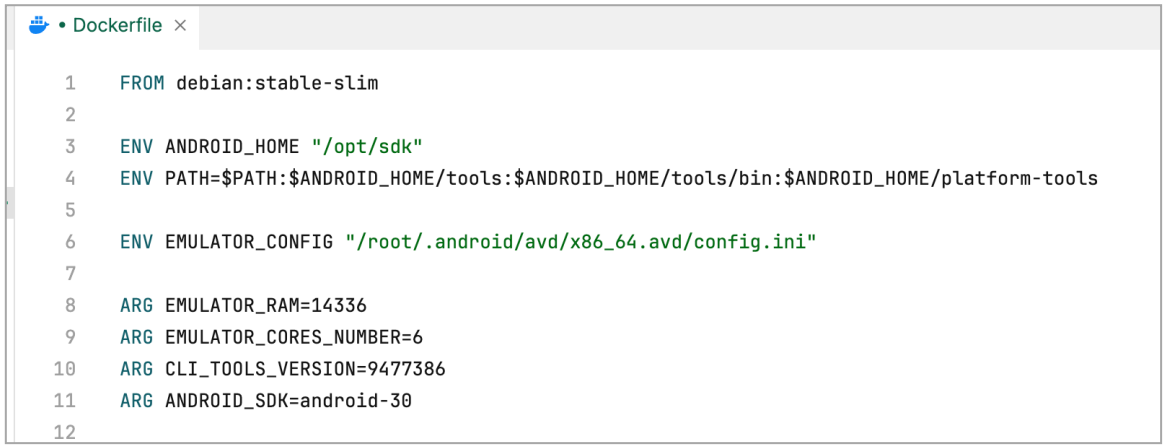
```
FROM debian:bullseye
```

Определяем ENV домашней директории android

```
ENV ANDROID_HOME «/opt/sdk"

ENV PATH=$PATH:$ANDROID_HOME/tools:$ANDROID_HOME/tools/bin:$ANDROID_HOME/platform-tools

ENV EMULATOR_CONFIG "/root/.android/avd/x86_64.avd/config.ini"
```

Добавляем дефолтные значения ARG (аргументов)

```
ARG EMULATOR_RAM=14336
ARG EMULATOR_CORES_NUMBER=6
ARG CLI_TOOLS_VERSION=9477386
ARG ANDROID_SDK=android-30
```



Ставим зависимости для ОС и эмуляторов

```
RUN apt-get update && apt-get -y dist-upgrade && DEBIAN_FRONTEND=noninteractive apt-get
install -y --no-install-recommends \
   telnet openjdk-17-jdk bash git unzip wget \
   redir qemu-utils procps iproute2 \
   libx11-dev libpulse-dev libnss3 \
   libxcomposite-dev libxcursor-dev \
   libxi-dev libxtst-dev uuid-dev \
   libgl-dev libasound-dev libxcb1-dev && \
   apt-get -y autoremove && \
   apt-get clean autoclean && \
    rm -rf /var/lib/apt/lists/*
```

Ставим зависимости для ОС и эмуляторов

```
RUN apt-get update && apt-get -y dist-upgrade && DEBIAN_FRONTEND=noninteractive apt-get
install -y --no-install-recommends \

  telnet
  openjdk-17-jdk
  bash
  git
  unzip
  wget
  redir
  qemu-utils
  procps
  iproute2
. . .
    apt-get -y autoremove && \
    apt-get clean autoclean && \
    rm -rf /var/lib/apt/lists/*
```

Устанавливаем необходимые зависимости для нашей ОС и наших будущих эмуляторов

```
RUN apt-get update && apt-get -y dist-upgrade && DEBIAN_FRONTEND=noninteractive apt-get
install -y --no-install-recommends \
...
  libx11-dev \
  libpulse-dev \
  libnss3 \
  libxcomposite-dev \
  libxcursor-dev \

...
    apt-get -y autoremove && \
    apt-get clean autoclean && \
    rm -rf /var/lib/apt/lists/*
```

Устанавливаем необходимые зависимости для нашей ОС и наших будущих эмуляторов

```
RUN apt-get update && apt-get -y dist-upgrade && DEBIAN_FRONTEND=noninteractive apt-get
install -y --no-install-recommends \
. . .
    libxi-dev
    libxtst-dev
    uuid-dev \
. . .
   apt-get -y autoremove && \
   apt-get clean autoclean && \
   rm -rf /var/lib/apt/lists/*
```

Устанавливаем необходимые зависимости для нашей ОС и наших будущих эмуляторов

```
RUN apt-get update && apt-get -y dist-upgrade && DEBIAN_FRONTEND=noninteractive apt-get
install -y --no-install-recommends \
. . .
    libgl-dev
    libasound-dev
    libxcb1-dev &&
. . .
    apt-get -y autoremove && \
   apt-get clean autoclean && \
    rm -rf /var/lib/apt/lists/*
```

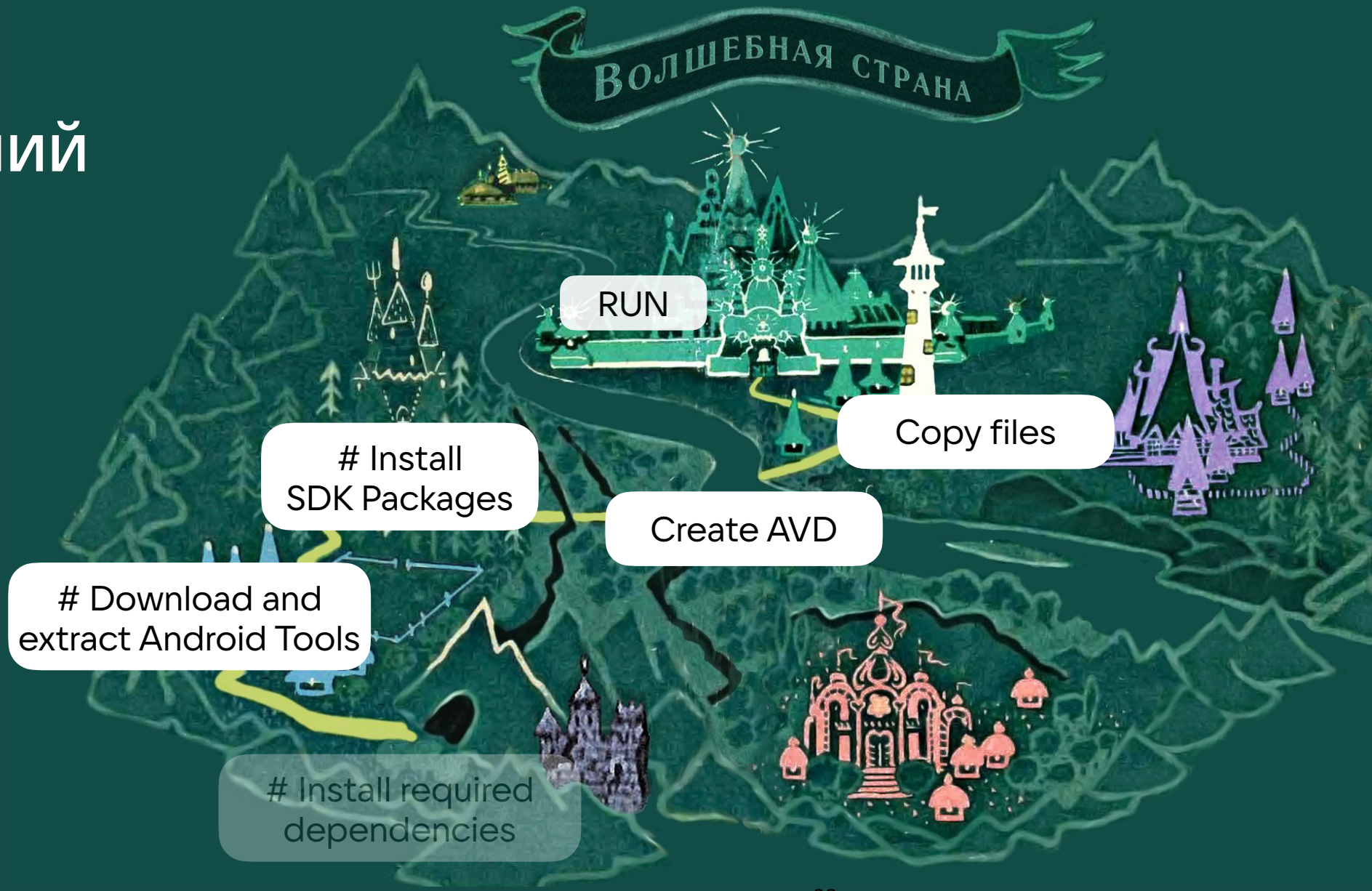
Загружаем и устанавливаем андроид тул для линукса

Посмотреть актуальную версию можно здесь - https://developer.android.com/studio#downloads

Посмотреть релиз ноутс можно здесь - https://developer.android.com/studio/releases/platform-tools#revisions

Посмотреть все версии https://mirrors.cloud.tencent.com/AndroidSDK/ https://androidsdkmanager.azurewebsites.net/SDKPlatform

Масштаб приключений



Загружаем и устанавливаем андроид тул для линукса

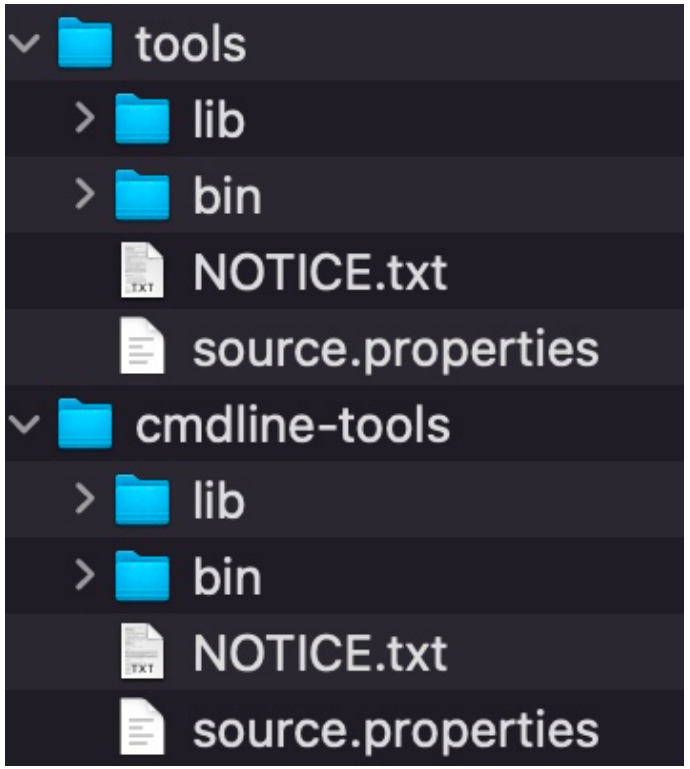
```
# Download and extract Android Tools
ARG CLI_TOOLS_VERSION=8512546
RUN wget -q
https://dl.google.com/android/repository/commandlinetools-linux-${CLI_TOOLS_VERSION}_latest.zip
-O /tmp/tools.zip && \
    mkdir -p ${ANDROID_HOME} && \
    unzip -qq /tmp/tools.zip -d ${ANDROID_HOME} && \
    mv ${ANDROID_HOME}/cmdline-tools ${ANDROID_HOME}/tools && \
    rm -v /tmp/tools.zip
```

NB

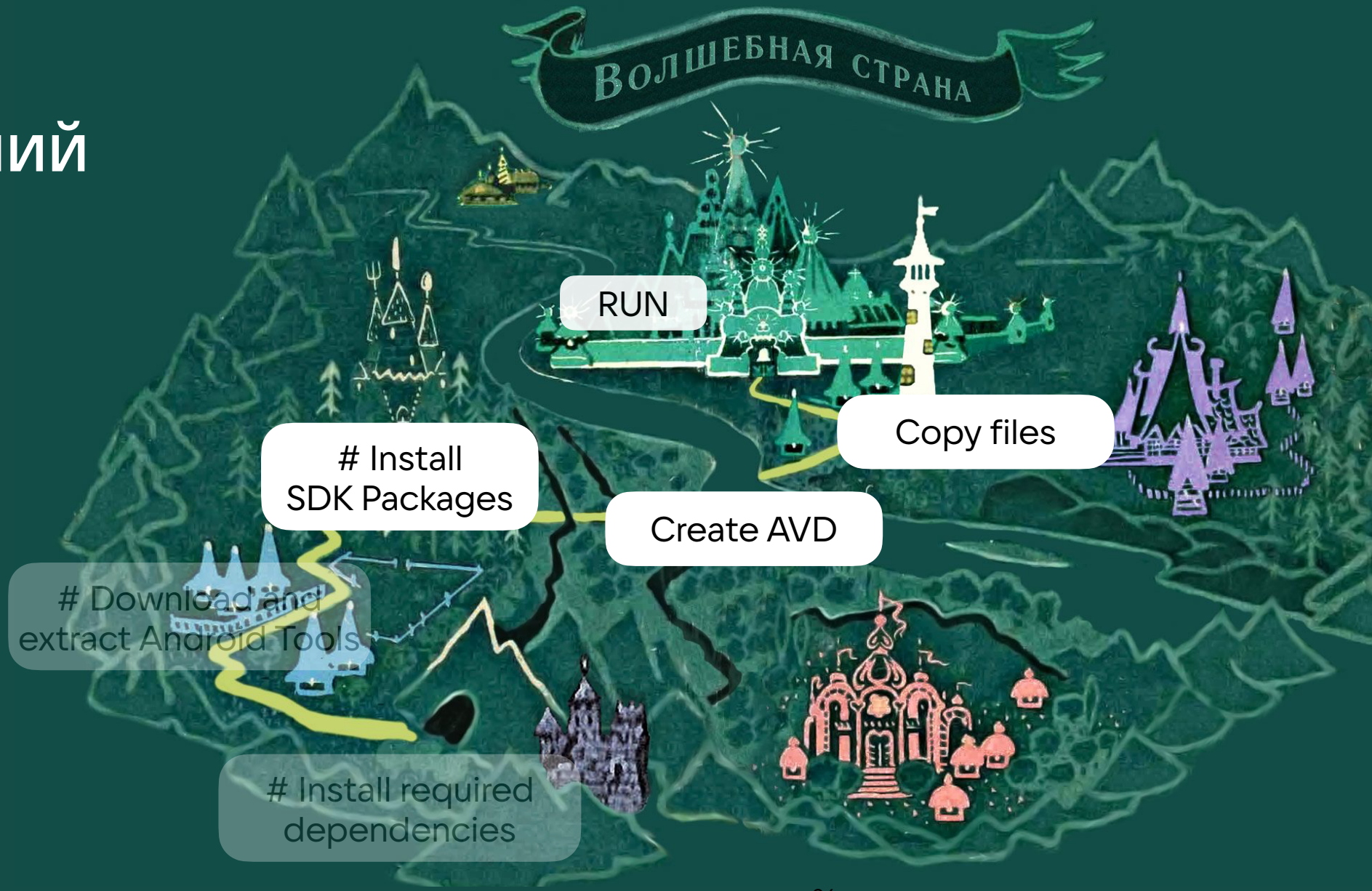
В commandlinetools 6825553 от сентября 2020 года была смена нейминга папки

Переименовать папку можно командой

```
mv ${ANDROID_HOME}/cmdline-tools ${ANDROID_HOME}/tools
```



Масштаб приключений



Подготовка образа для сборки

```
mkdir -p ~/.android/ && touch ~/.android/
repositories.cfg && \
```

Создаем директорию
и конфиг файл

```
yes | ${ANDROID_HOME}/tools/bin/sdkmanager
--sdk_root=${ANDROID_HOME} --licenses && \
```

Принимаем
все лицензии

```
${ANDROID_HOME}/tools/bin/sdkmanager --
sdk_root=${ANDROID_HOME} --update && \
```

Устанавливаем
обновления

Подготовка образа для сборки

```
${ANDROID_HOME}/tools/bin/sdkmanager --
sdk_root=${ANDROID_HOME} emulator "system-
images;${PLATFORM};google_apis;x86_64"
platform-tools "platforms;${PLATFORM}" && \
```

Выбираем необходимую нам версию эмулятора, наличие google API и версию

Подготовка образа для сборки

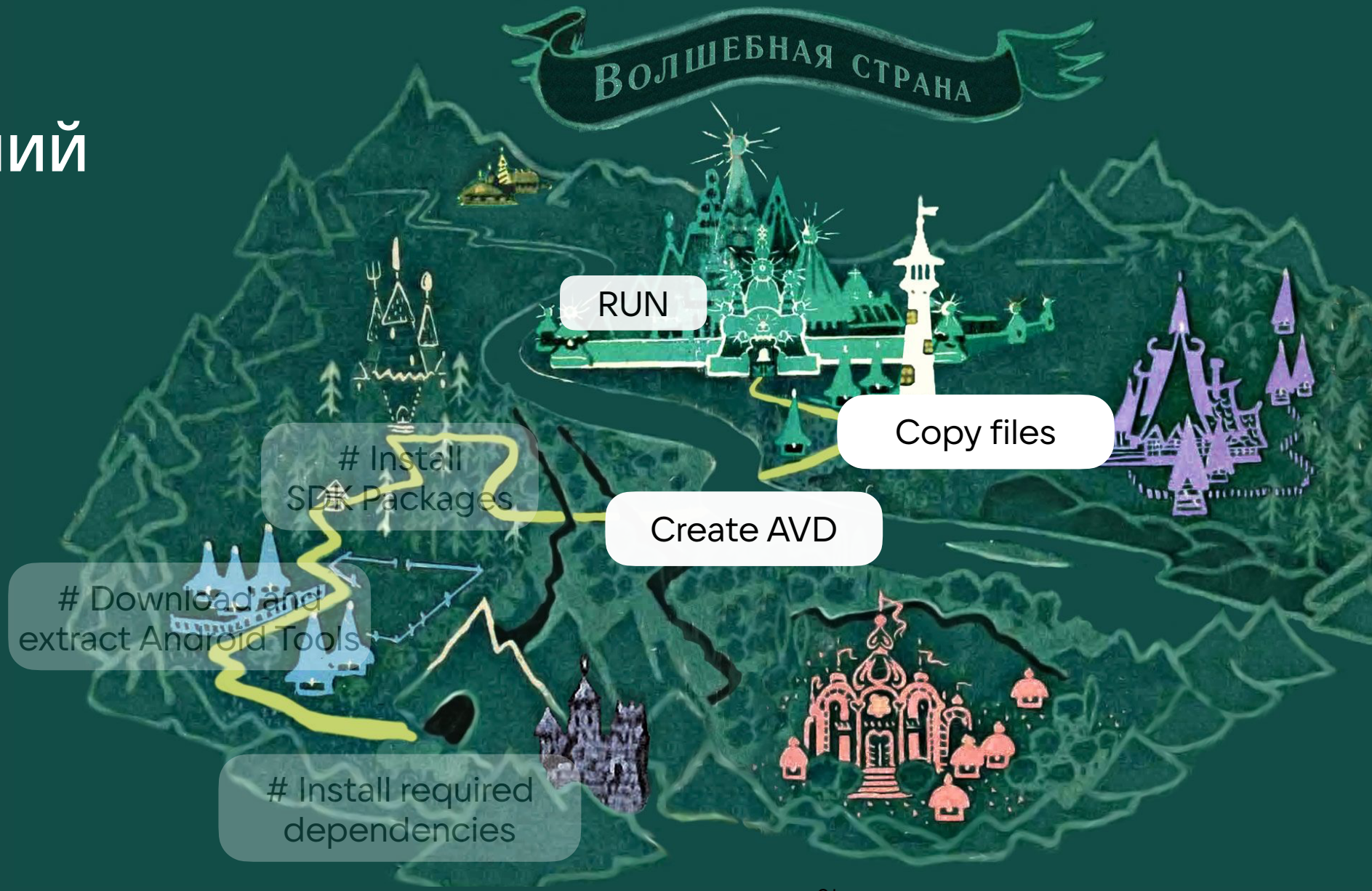
```
qemu-img convert -O qcow2 -c
${ANDROID_HOME}/system-images/${PLATFORM}/
google_apis/x86_64/userdata.img $
{ANDROID_HOME}/system-images/${PLATFORM}/
google_apis/x86_64/userdata.qcow2 && \

mv ${ANDROID_HOME}/system-images/
${PLATFORM}/google_apis/x86_64/
userdata.qcow2 ${ANDROID_HOME}/system-
images/${PLATFORM}/google_apis/x86_64/
userdata.img
```

Сжимаем образ userdata.img в QCOW2 для экономии места и перемещаем его

Все в сборе

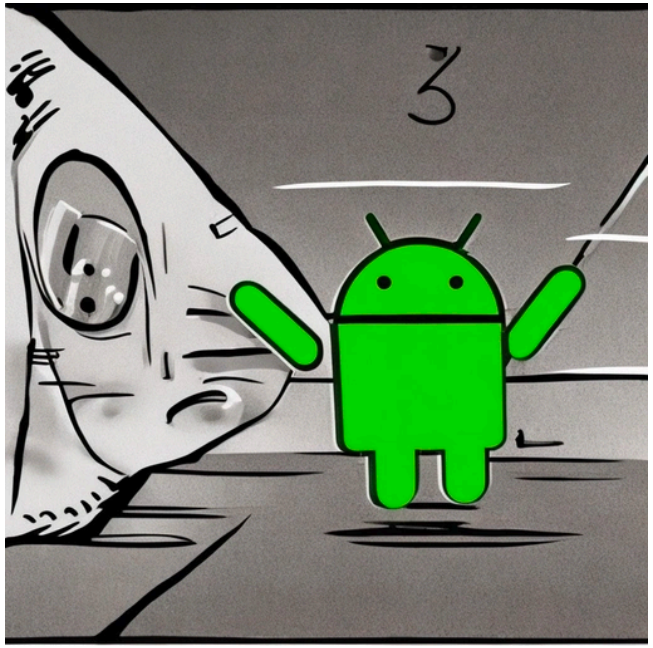
```
RUN mkdir -p ~/.android/ && touch ~/.android/repositories.cfg && \
    yes | ${ANDROID_HOME}/tools/bin/sdkmanager --sdk_root=${ANDROID_HOME} --licenses && \
    ${ANDROID_HOME}/tools/bin/sdkmanager --sdk_root=${ANDROID_HOME} --update && \
    ${ANDROID_HOME}/tools/bin/sdkmanager --sdk_root=${ANDROID_HOME} emulator "system-images;$
{ANDROID_SDK};google_apis;x86_64" platform-tools "platforms;${ANDROID_SDK}" && \

    # Compress downloaded system.img
    echo "Converting system.img to qcow2" && \
    qemu-img convert -O qcow2 -c ${ANDROID_HOME}/system-images/${ANDROID_SDK}/google_apis/x86_64/
system.img ${ANDROID_HOME}/system-images/${ANDROID_SDK}/google_apis/x86_64/system.qcow2 && \
    mv ${ANDROID_HOME}/system-images/${ANDROID_SDK}/google_apis/x86_64/system.qcow2 $
{ANDROID_HOME}/system-images/${ANDROID_SDK}/google_apis/x86_64/system.img && \

    # Compress downloaded userdata.img
    echo "Converting userdata.img to qcow2" && \
    qemu-img convert -O qcow2 -c ${ANDROID_HOME}/system-images/${ANDROID_SDK}/google_apis/x86_64/
userdata.img ${ANDROID_HOME}/system-images/${ANDROID_SDK}/google_apis/x86_64/userdata.qcow2 && \
    mv ${ANDROID_HOME}/system-images/${ANDROID_SDK}/google_apis/x86_64/userdata.qcow2 $
{ANDROID_HOME}/system-images/${ANDROID_SDK}/google_apis/x86_64/userdata.img
```

Масштаб приключений



Создаем эмулятор

```
RUN RUN echo | ${ANDROID_HOME}/tools/bin/avdmanager
        create avd --name "x86_64"  --package "system-images;$
        {ANDROID_SDK};google_apis;x86_64" --tag google_apis
```



Масштаб приключений

Создаем start.sh, liveness.py и config.ini в корне и копируем в необходимую директорию

```
COPY start.sh /opt/sdk/start.sh
COPY liveness.py /opt/liveness.py
COPY config.ini ${EMULATOR_CONFIG}
```

Масштаб приключений

Выдаем права для запуска

```
RUN echo image.sysdir.1=system-images/${ANDROID_SDK}/google_apis/x86_64/ >> $
    {EMULATOR_CONFIG} && \
    echo hw.cpu.ncore=${EMULATOR_CORES_NUMBER} >> ${EMULATOR_CONFIG} &&  \
    echo hw.ramSize=${EMULATOR_RAM} >> ${EMULATOR_CONFIG} &&\
    sed -i 's:image.sysdir.1=sdk/:image.sysdir.1=:g' "/root/.android/avd/x86_64.avd/
   config.ini" && \
    groupadd kvm && useradd -d /home/app -G kvm -m -r app && \
    chmod +x /opt/sdk/start.sh && \
    sed -i "s:!platform!:${ANDROID_SDK}:" /opt/sdk/start.sh && \
    find "${HOME}/.android/avd" -name *lock -exec rm {} \;
```

запускаем файлик

```
CMD /opt/sdk/start.sh
```

Переходим к start.sh

В скрипте лежат все необходимые команды для запуска эмулятора, после его создания из докерфайла

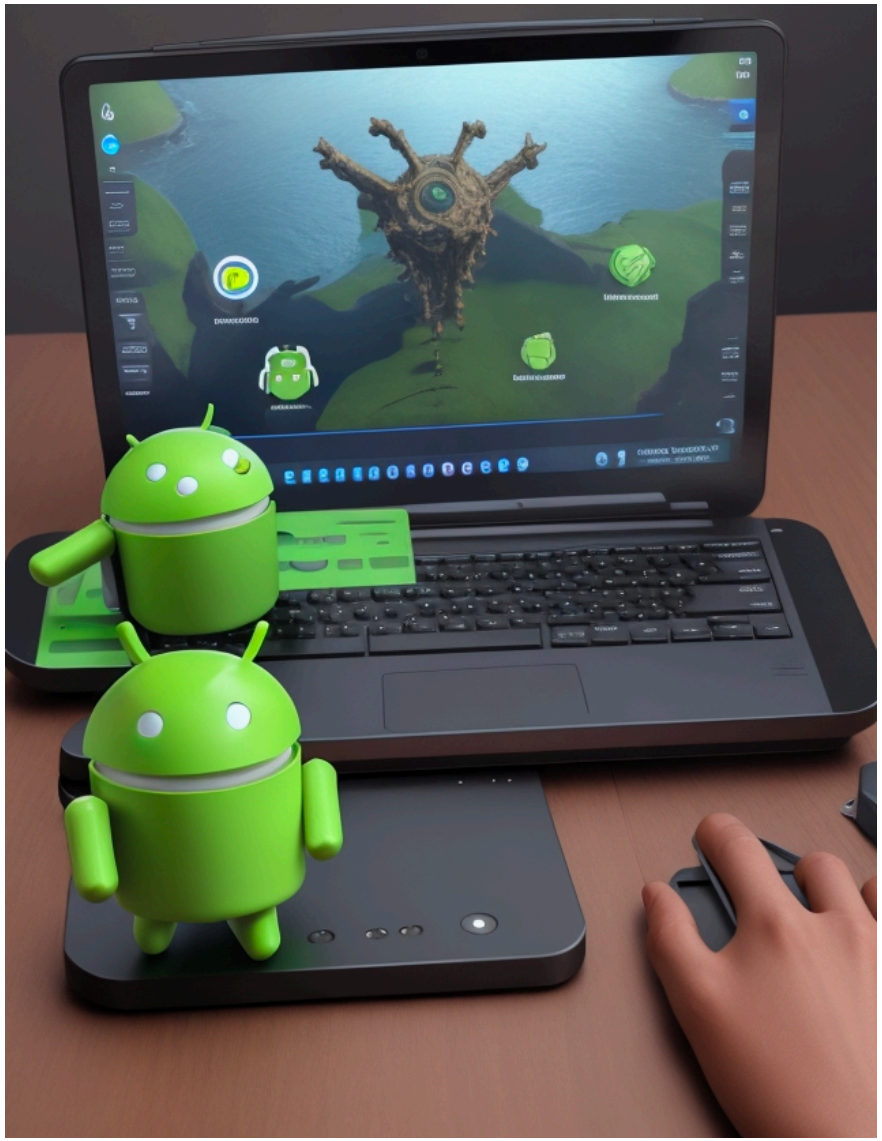
Переходим к Start.sh

Определяем необходимые для нас параметры

```
LANG=en_US.UTF-8
LANGUAGE=en_US:en
LC_ALL=en_US.UTF-8
ANDROID_HOME=/opt/sdk
PLATFORM=!platform!
EMULATOR_ROOT="${ANDROID_HOME}/emulator"
PATH="$PATH:${ANDROID_HOME}/tools:${ANDROID_HOME}/platform-tools:
${ANDROID_HOME}/tools/bin:${ANDROID_HOME}/emulator"
```

Экспортируем необходимые для нас пути библиотек

```
# Export library paths
export ANDROID_SDK_ROOT=${ANDROID_HOME}
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:${EMULATOR_ROOT}/lib64/qt/lib:${EMULATOR_ROOT}/
lib64/libstdc++:${EMULATOR_ROOT}/lib64:${EMULATOR_ROOT}/lib64/gles_swiftshader
export LIBGL_DEBUG=verbose
```



Определяем порты

Допустим, порты 10000 и 10001

```
# Emulator options
CONSOLE_PORT=10000
ADB_PORT=10001
```

Настраиваем необходимые нам аргументы при запуске эмулятора

```
EMULATOR_CONFIG="$HOME/.android/avd/x86_64.avd/
config.ini"
if [[ ! -d "$HOME/.android/avd/" ]] ; then
    echo | ${ANDROID_HOME}/tools/bin/avdmanager \
      create avd \
      --name "x86_64" \
      --package "system-images;$
{PLATFORM};google_apis;x86_64" \
      --tag google_apis
```

Если ничего больше
не требуется,
то пишем `fi`

Кастомные значения эмулятора

Кастомные значения эмулятор

Полное описание всех значений
в EMULATOR_CONFIG

Доклад Даниила Смирнова про наши приключения

https://heisenbug.ru/talks/4cbf30da4f9c48ea9f76cf3abfec76f7/

Кастомные значения для эмулятора

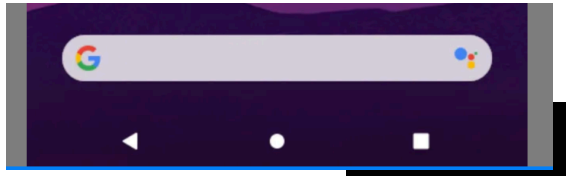
```
sed -i
's:image.sysdir.1=sdk/:image.sysdir.1=:g' "$
{EMULATOR_CONFIG}"
```

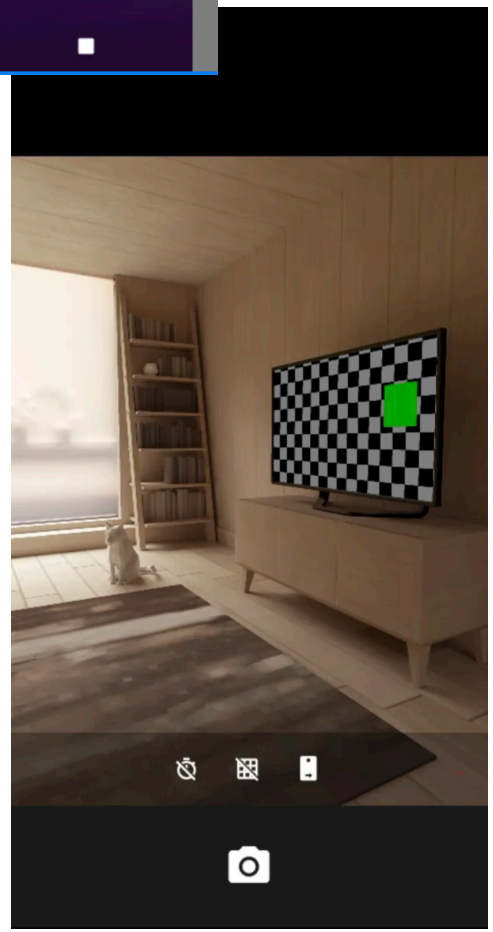
Исправляем путь образа

```
echo 'skin.name=480x980' >> «$
{EMULATOR_CONFIG}"

echo 'hw.lcd.height =480' >> «$
{EMULATOR_CONFIG}»

echo 'hw.lcd.width=980' >> «$
{EMULATOR_CONFIG}"
```

Увеличиваем базовое разрешение эмулятора,

Кастомные значения для эмулятора





Активируем наэкранные кнопки,

```
echo 'hw.mainKeys=no' >> "${EMULATOR_CONFIG}"
```

Активируем заднюю камеру,

```
echo 'hw.camera.back=virtualscene' >> "${EMULATOR_CONFIG}"
```

Кастомные значения для эмулятора

Активируем по той же логике фронталку, чтобы она точно была

```
echo 'hw.camera.front=emulated'
>> «${EMULATOR_CONFIG}"
```

Активируем ускорение на GPU *

если возможно

```
echo 'hw.gpu.enabled=yes'
>> "${EMULATOR_CONFIG}"
```

Кастомные значения для эмулятора

Активируем карту памяти, но делаем ее маленького размера

```
echo 'sdcard.size=256' >> «${EMULATOR_CONFIG}"
```

Включаем PlayStore

```
echo 'PlayStore.enabled=true' >> "${EMULATOR_CONFIG}"
```

Shell in emulator ▾ in experement-phone-33-6cfcz8w97

```
avd.id=<build>
avd.ini.encoding=UTF-8
avd.name=<build>
disk.cachePartition=yes
disk.cachePartition.size=66MB
disk.dataPartition.path=<temp>
disk.systemPartition.size=0
disk.vendorPartition.size=0
fastboot.forceChosenSnapshotBoot=no
fastboot.forceColdBoot=no
fastboot.forceFastBoot=yes
hw.accelerometer=yes
hw.arc=no
hw.arc.autologin=no
hw.audioInput=yes
hw.audioOutput=yes
hw.battery=yes
hw.dPad=yes
hw.display1.density=0
hw.display1.flag=0
hw.display1.height=0
hw.display1.width=0
hw.display1.xOffset=-1
hw.display1.yOffset=-1
hw.display2.density=0
hw.display2.flag=0
```

Все значения записываются в файлик, доступный по ссылке

```
cat root/.android/avd/
x86_64.avd/config.ini
```

Делаем возможным передачу кастомных значений в конфиг эмулятора

```
CONFIG="$HOME/.android/avd/x86_64.avd/config.ini"
CONFIGTMP=${CONFIG}.tmp

if [ -n "$ANDROID_CONFIG" ];
then
  IFS=';' read -ra OPTS <<< "$ANDROID_CONFIG"
  for OPT in "${OPTS[@]}"; do
    IFS='=' read -ra KV <<< "$OPT"
    KEY=${KV[0]}
    VALUE=${KV[1]}
    mv ${CONFIG} ${CONFIGTMP}
    cat ${CONFIGTMP} | grep -v ${KEY}= > ${CONFIG}
    echo ${OPT} >> ${CONFIG}
  done
fi
```

Настраиваем размера файла подкачки для эмулятора

```
# Set heap size to max if not defined in the
environment
if [[ -z "${EMULATOR_HEAP_SIZE}" ]] ; then
    EMULATOR_HEAP_SIZE=2048
fi
sed -i -E "s:(vm\.heapSize)=(.*):\1=$
{EMULATOR_HEAP_SIZE}:" "${EMULATOR_CONFIG}"
```

Почитать про него можно
по ссылке

Если вы не хотите передавать эти параметры

В нашем конфиг файле все эти значения уже вписаны

config.ini

```
avd.id=x86_64
avd.name=x86_64
avd.ini.encoding=UTF-8
disk.cachePartition=yes
disk.cachePartition.size=66MB
disk.dataPartition.path=<temp>
disk.systemPartition.size=0
disk.vendorPartition.size=0
fastboot.forceChosenSnapshotBoot=no
fastboot.forceColdBoot=no
fastboot.forceFastBoot=yes
hw.accelerometer=yes
hw.arc=no
hw.arc.autologin=no
hw.audioInput=yes
hw.audioOutput=yes
hw.battery=yes
hw.dPad=yes
hw.gltransport=pipe
hw.gltransport.asg.dataRingSize=32768
hw.gltransport.asg.writeBufferSize=1048576
hw.gltransport.asg.writeStepSize=4096
hw.gltransport.drawFlushInterval=800
hw.gps=yes
hw.gpu.mode=auto
hw.gsmModem=yes
hw.gyroscope=yes
hw.initialOrientation=portrait
hw.keyboard=no
```

Дефолтные значения эмулятора

Откуда мы взяли эти значения?

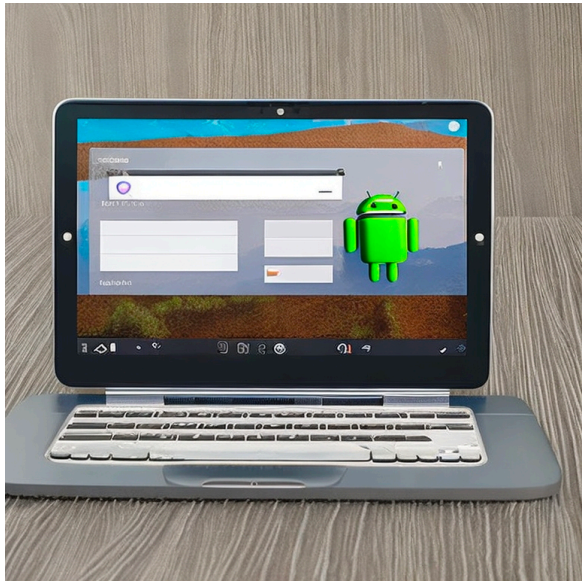
Heisenbug 2022

https://heisenbug.ru/talks/2f486c767b6b99e6a9a2188ace7460d9/

Настраиваем дефолтные опции для эмулятора,

Аргументы для запуска эмулятора из командой строки можно почерпнуть здесь - https://developer.android.com/studio/run/emulator-commandline

```
# Set default emulator options if not defined in the environment
if [[ -z "${EMULATOR_OPTS}" ]] ; then
    EMULATOR_OPTS="-screen multi-touch -no-boot-anim -noaudio -netfast -verbose
    -skip-adb-auth -no-snapshot -no-snapstorage -camera-back virtualscene
    -camera-front emulated -partition-size 14336 -version -selinux disabled"

fi
# Add any extra opts
EMULATOR_OPTS+=" ${EXTRA_EMULATOR_OPTS}"
```



Дефолтное значение оперативной памяти эмулятора

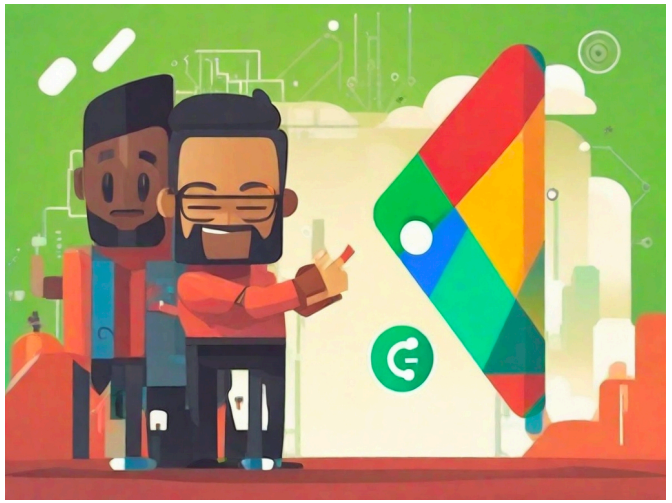
```
# Set RAM configuration if provided in the environment
if [[ -z "${EMULATOR_RAM_SIZE}" ]] ; then
    EMULATOR_RAM_SIZE=14336
fi
sed -i -E "s:(hw\.ramSize)=(.*):\1=${EMULATOR_RAM_SIZE}:" "${EMULATOR_CONFIG}"
```

Определяем количество ядер

```
# Set CPU configuration
if [[ -z "${EMULATOR_NUM_CORE}" ]] ; then
    EMULATOR_NUM_CORE=6
fi
sed -i -E "s:(hw\.cpu\.ncore)=(.*):\1=$
{EMULATOR_NUM_CORE}:" "${EMULATOR_CONFIG}"
```

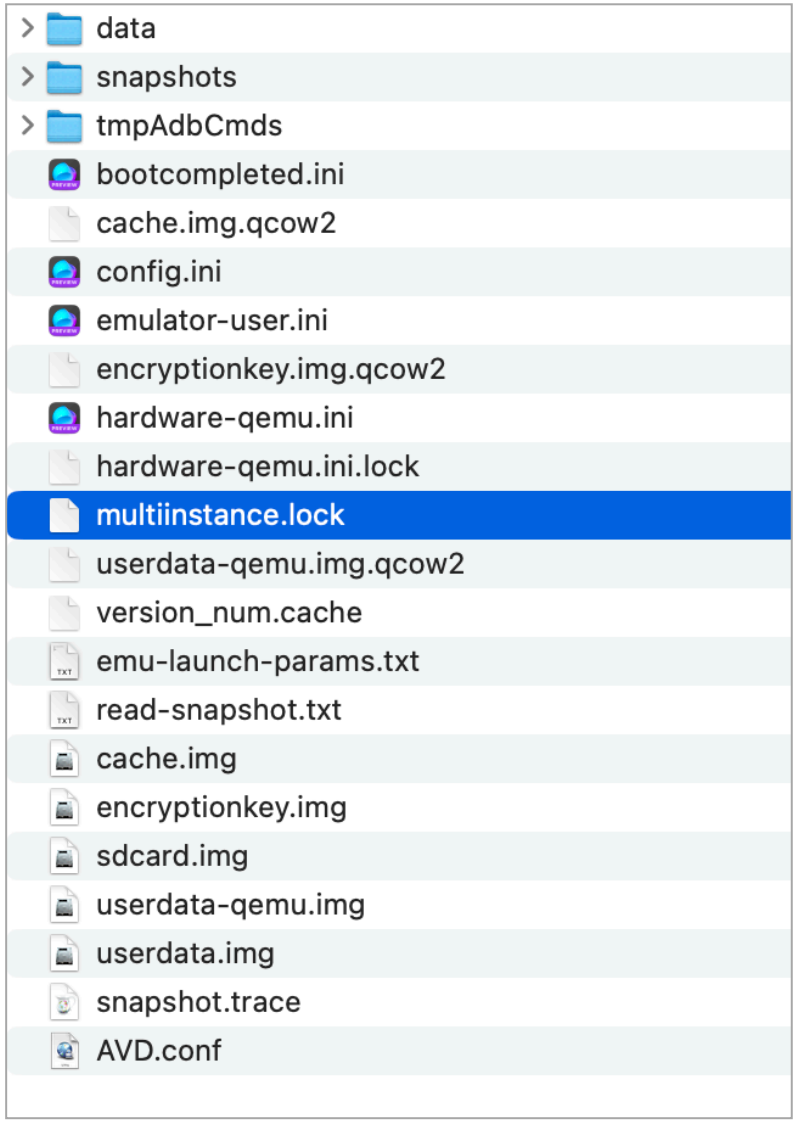
Конфигурируем доступность play store

```
# Configure Play Store
if [[ "${EMULATOR_ENABLE_PLAYSTORE,,}" == "true" ]] ; then
    grep -q '^PlayStore' "${EMULATOR_CONFIG}" && \
    sed -i -E "s:(PlayStore\.enabled)=(.*):\1=true:" "${EMULATOR_CONFIG}" || echo
'PlayStore.enabled=true' >> "${EMULATOR_CONFIG}"
fi
```



Удаляем lock файлы

```
find "${HOME}/.android/avd"
-name *lock -exec rm {} \;
```



Поднимаем эмулятор

```
# Start the emulator
${EMULATOR_ROOT}/qemu/linux-x86_64/qemu-system-x86_64-headless \
  -avd x86_64 \
  -gpu auto \
  -no-window \
  -timezone Europe/Moscow \
  -ports ${CONSOLE_PORT},${ADB_PORT} \
  ${EMULATOR_OPTS} &
```

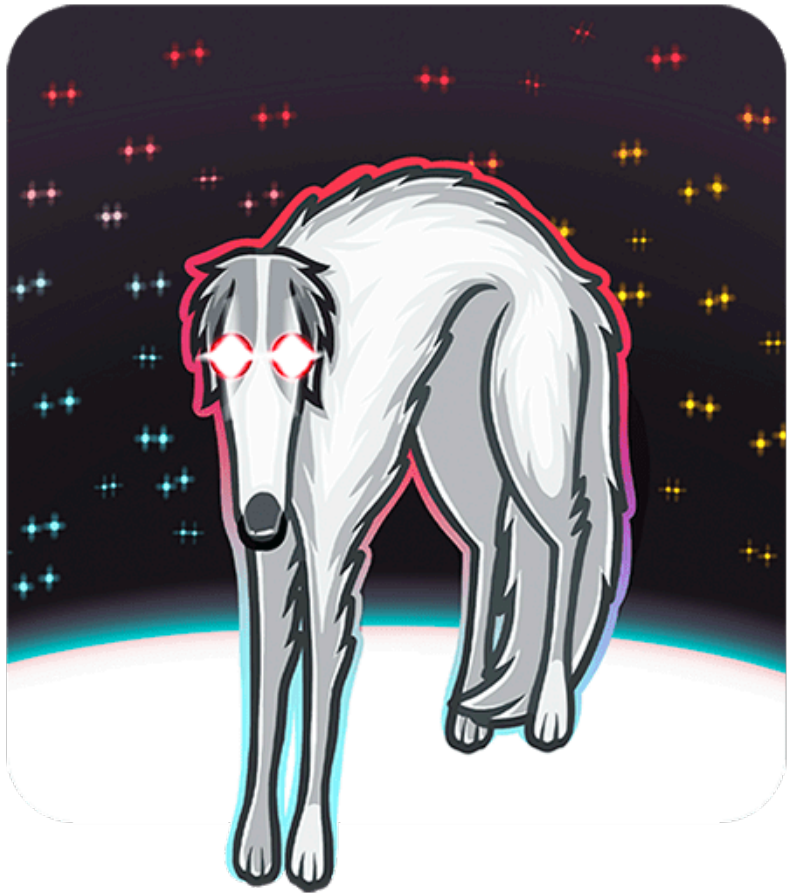
- Название
  - -avd x86_64 \
- GPU
  - -gpu auto \
- Режим no window
  - Для ускорения
- Указываем таймзону
  - -timezone Europe/Moscow \
  - Как выглядит время в TZ- https://en.wikipedia.org/wiki/List_of_tz_database_time_zones
- Указываем открытые порты
  - -ports ${CONSOLE_PORT},${ADB_PORT} \
- Добавляем еще параметры, указанные выше

Настраиваем наш эмулятор заранее

У нас настроены следующие вещи, которые мы делаем под рутовым пользователем через команду adb shell su 0.

Т.к. передается много аргументов, то используем ' * ', для исполнения последовательно используем &&

```
echo "Setup emulator settings throw shell manipulation."
adb shell su 0 'settings put system system_locales ru-RU &&
settings put secure immersive_mode_confirmations confirmed &&
settings put global window_animation_scale 0.0 &&
settings put global transition_animation_scale 0.0 &&
settings put global animator_duration_scale 0.0 &&
settings put secure long_press_timeout 1800 &&
settings put secure show_ime_with_hard_keyboard 1 &&
settings put secure spell_checker_enabled 0 &&
settings put secure autofill_service null &&
settings put global hidden_api_policy 1 &&
settings put system pointer_location 1 &&
settings put system show_touches 1 &&
echo "chrome --disable-fre --no-default-browser-check --no-
first-run" > /data/local/tmp/chrome-command-line &&
am set-debug-app —persistent com.android.chrome'

adb logcat -G 16M
sleep 2
echo "Rebooting emulator after shell manipulation."
adb reboot
adb wait-for-device
```



Настройка эмулятора

```
settings put system
system_locales ru-RU
```

Русская локаль
(по дефолту английская)

```
settings put secure immersive_mode_confirmations
confirmed
```

Принятие полноэкранных
уведомлений

```
settings put global window_animation_scale 0.0
settings put global transition_animation_scale 0.0
settings put global animator_duration_scale 0.0
```

Отключение
анимации в системе

Настройка эмулятора

```
settings put secure
long_press_timeout 1800
```

Настраиваем длительность лонглика, чтобы избежать некорректной трактовки

```
settings put secure
show_ime_with_hard_keyboard 1
```

Настраиваем клавиатуру

```
settings put secure
spell_checker_enabled 0
```

Отключаем механизм озвучивания

Настройка эмулятора

`settings put secure autofill_service null`	Отключаем автозаполнение
`settings put global hidden_api_policy 1`	Открываем доступ к скрытому апи
`settings put system pointer_location 1`	Активируем отображение координат на экране

Настройка эмулятора

```
settings put system show_touches 1
```

Активируем отображение нажатий

```
echo "chrome --disable-fre --no-default-browser-
check --no-first-run" > /data/local/tmp/chrome-
command-line &&
am set-debug-app —persistent com.android.chrome
```

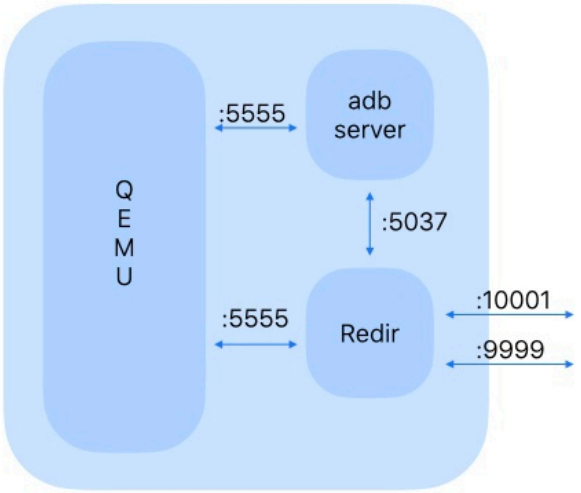
Убираем онбординг браузера chrome

```
adb logcat -G 16M
```

Увеличиваем размер логката до максимальных 16 мегабайт

Настраиваем редиректы

```
ip=$(ip addr list eth0|grep "inet "|cut -d' ' -f6|cut -d/ -f1)

redir –laddr=$ip –lport=9999 –caddr=127.0.0.1 –cport=5037 &
redir –laddr=$ip –lport=10000 –caddr=127.0.0.1 –cport=10000 &
redir –laddr=$ip –lport=10001 –caddr=127.0.0.1 –cport=10001 &
```



Эмулятор

Для чего нужен	Внутри контейнера	Снаружи контейнера
Порт для подсоединения к эмулятору	:5555	:10001
Adb server для исполнения стартовых комманд эмулятора	:5037	:9999

Ожидаем процесс

```
wait ${EMULATOR_PID}
echo "Emulator preparation finished"
```

Как можно проверить введенные значения?

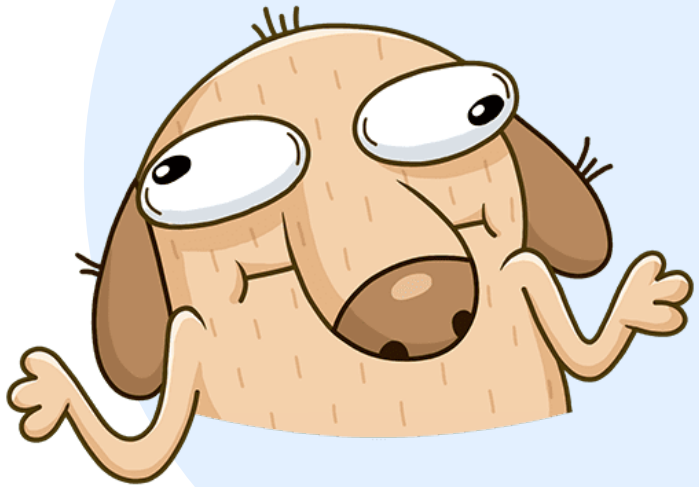
Получаем готовый набор команд. Дополнительными командами через адб в файле start.sh мы можем убедиться, что наши команды выполнены и все работает корректно. Например,

```
require 0.0 adb shell "settings get
global animator_duration_scale"
```

Переходим к liveness.py

Переходим к liveness.py

- Эмулятор живущий долго - может деградировать. Нам необходимо отслеживать такие ситуации.
- Мы путем проб и ошибок нашли лайвнес пробу, которая нам нужна и отвечает на наш вопрос



Настройка liveness.py

liveness.py

```
1  from telnetlib import Telnet
2  import logging
3  import subprocess
```

Нам необходимы следующие библиотеки

```
from telnetlib import Telnet
import logging
import subprocess
```

Настройка liveness.py

```
logging.basicConfig(level=logging.INFO)


1 usage
def check_by_telnet():
    f = open('/root/.emulator_console_auth_token')
    token = f.readline()
    f.close()
    logging.info('Auth token acquired')
```

Обозначаем уровень логирования

```
logging.basicConfig(level=logging.INFO)
```

Запрашиваем токен авторизации

```
f = open('/
root/.emulator_console_auth_token')
token = f.readline()
f.close()
logging.info('Auth token acquired')
```

Настройка liveness.py

```
telnet = Telnet('127.0.0.1', 10000)
expectation = telnet.expect([b'OK'], timeout=1)
if not expectation[1]:
    logging.error('Cant connect to emulator')
    exit(1)
logging.info('Connected to emulator')
```

Подключаемся к девайсу через telnet


```
telnet = Telnet('127.0.0.1', 10000)
expectation = telnet.expect([b'OK'],
timeout=1)
if not expectation[1]:
    exit(1)
logging.info('Connected to emulator')
```

Настройка liveness.py

```
auth_line = ('auth ' + token + '\n').encode('ascii')
telnet.write(auth_line)
expectation = telnet.expect([b'OK'], timeout=1)
if not expectation[1]:
    logging.error('Cant auth in emulator')
    exit(1)
logging.info('Authed in emulator')
```

Авторизуемся на эмуляторе для получения необходимой информации

```
auth_line = ('auth ' + token +
'\n').encode('ascii')
telnet.write(auth_line)
expectation = telnet.expect([b'OK'],
timeout=1)
if not expectation[1]:
    exit(1)
logging.info('Authed in emulator')
```

Настройка liveness.py

```
status_line = 'avd status\n'.encode('ascii')
telnet.write(status_line)
expectation = telnet.expect([b'virtual device is running'], timeout=1)
if not expectation[1]:
    logging.error('Emulator is not running')
    exit(1)
logging.info('telnet is ok')
```

Проверяем статус устройства

```
status_line = 'avd
status\n'.encode('ascii')
telnet.write(status_line)
expectation =
telnet.expect([b'virtual device
is running'], timeout=1)
if not expectation[1]:
    exit(1)
logging.info('telnet is ok')
```

Настройка liveness.py

```
subprocess.Popen('adb shell getprop sys.boot_completed | grep -q "1"', shell=True).wait(10)
logging.info('boot is ok')
```

Получаем статус sys.boot_completed, который говорит, что загрузка завершена

```
subprocess.Popen('adb shell getprop sys.boot_completed | grep -q "1"', shell=True).wait(3)
logging.info('boot is ok')
```

Использование liveness.py в Kubernetes

livenessProbe:

exec:
command:
- python3
- /opt/liveness.py
initialDelaySeconds: 10
timeoutSeconds: 1
periodSeconds: 10
successThreshold: 1
failureThreshold: 3

Где мы

1. Для чего нам нужны эмулятор
2. Собираем наш эмулятор
3. Вносим изменения в эмулятор и сокращаем вес docker образа
4. Полезные ссылки

Вносим изменения и сохраняем

Что делаем после внесения изменений?

Проверяем, что все корректно и переходим ко второму этапу.

1. Собираем получившееся через docker
2. Переходим в терминал
3. через cd спускаемся в директорию, где мы сохранили докер образ и файлы

Что делаем после внесения изменений?

Проверяем, что все корректно и переходим ко второму этапу.

1. Выполняем команду docker build -t emulator-30 .
   - docker build отвечает за билд докер образа
   - -t - tag
   - emulator-30 - название эмулятора
2. После того, как все собралось мы можем запустить наш новый эмулятор
   - docker run -d —name android -p 10001:10001 -p 10000:10000 —device=/dev/kvm -v /root/snapshots:/snapshots

Запускаем эмулятор

- Для того, чтобы посмотреть все образы на машине можно использовать команду Docker images
- Итак, мы собрали новый образ эмулятора, запустили его и убедились, что все хорошо и все работает.
- Но что делать, если мы хотим внести какие-то изменения в текущий образ, а не только поднимать совершенно новый?
- Для этого мы можем воспользоваться docker commit и сохранить изменения

Docker commit

- Для того, чтобы посмотреть все образы на машине можно использовать команду Docker images
- Итак, мы собрали новый образ эмулятора, запустили его и убедились, что все хорошо и все работает.

- Но что делать, если мы хотим внести какие-то изменения в текущий образ, а не только поднимать совершенно новый?
- Для этого мы можем воспользоваться docker commit и сохранить изменения

Docker commit

- После того, как мы внесли изменения в наш эмулятор создаем коммит
  - Используем команду docker commit DOCKER_ID emulator-30.

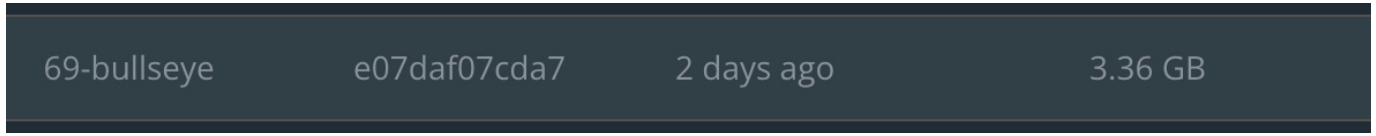
androidsdk/android-30	latest	719db0146c62	about 2 years ago	5.67 GB

Снижаем вес образа

- Есть альтернативные способы для внесения изменения только в основной файл с данными эмулятор

Savedata.sh

```
docker exec -w /data android-%emulator_version% /usr/bin/qemu-img commit userdata-qemu.img.qcow2
docker exec -w /data android-%emulator_version% /usr/bin/qemu-img commit encryptionkey.img.qcow2
docker stop $(docker ps -a -q)
docker rm $(docker ps -a -q)
qemu-img convert -O qcow2 /data/android-%emulator_version%/userdata-qemu.img /data/android-
%emulator_version%/userdata-qemu.img.qcow2
cp /data/android-%emulator_version%/userdata-qemu.img.qcow2 .
cp /data/android-%emulator_version%/encryptionkey.img .
```



Полезные ссылки

Полезные ссылки

Ссылка на репозиторий VK.com в github

https://github.com/VKCOM/docker-emulator-android

Полезные ссылки

Если вам не требуется дополнительная кастомизация эмуляторов, то вы можете пропустить эти шаги и использовать уже готовые docker образы от команды google
https://console.cloud.google.com/artifacts/docker/android-emulator-268719/us/images/30-google-x64