Ручная сборка Arch ISO (Headless + WebUI)

Этап 1. Подготовка диска sdb на хосте tom_1

Выполняется на живом хосте tom_1 (IP: 192.168.1.72). Диск sdb выступает временным накопителем-донором. с доступом в интернет.

1.1 Создание рабочих директорий

Схема создаваемой структуры:

```
/home
└── eva
    ├── new_iso
    │   ├── boot
    │   └── etc
    │       └── fstab
    └── original_iso_image
        └── archlinux-x86_64.iso
```

#bash

```
# original_iso_image - для оригинального образа iso
mkdir -p ~/original_iso_image/
# new_iso - рабочай директория, где будем собирать новый образа iso
mkdir -p ~/new_iso/
```

Как это работает

- mkdir имя_папки — создает новую директорию.

Проаерка создания

#bash

```
# original_iso_image - для оригинального образа iso
ls -ld ~/original_iso_image/
# new_iso - рабочай директория, где будем собирать новый образа iso
ls -ld ~/new_iso/
```

Как это работает

- ls -ld имя_папки — выводит информацию о конкретной папке и её правах без показа содержимого.

1.2 Разметка виртуального диска sdb (будущий arch-flash)

Просмотр списка дисков и разделов

Самый наглядный способ увидеть все подключенные накопители, их размер и структуру:

#bash

```
lsblk
```

скрипт создаст правильную структуру для UEFI + BIOS:

sdb1 — BIOS Boot (1 МБ) → для загрузки старого BIOS.
sdb2 — EFI System (1 ГБ, FAT32) → для загрузки UEFI (ваш boot).
sdb3 — Linux root (Все оставшееся место, BTRFS)

Используйте один из следующих вариантов размеи дисков с использованием утилит fdisk или sfdisk

1.2.1 Использование утилиты fdisk

#bash

```
sudo fdisk /dev/sdb <<EOF
g
n
1

+1M
t
4
n
2

+1G
t
2
1
n
3



w
EOF
```

1.2.2 Использование утилиты sfdisk

#bash

```
sudo sfdisk /dev/sdb << EOF
label: gpt
device: /dev/sdb
unit: sectors

/dev/sdb1 : start=        2048, size=        2048,
type=21686148-6449-6E6F-744E-656564454649
/dev/sdb2 : start=        4096, size=     2097152, type=C12A7328-
F81F-11D2-BA4B-00A0C93EC93B
/dev/sdb3 : start=     2101248, size=           +,
type=0FC63DAF-8483-4772-8E79-3D69D8477DE4
EOF
```

Поверим

#bash

```
lsblk
```

1.3 Форматирование разделов с метками (LABEL)

1.3.1 Назначение метки ARCH_BOOT для UEFI-загрузчикаFAT32:

#bash

```
sudo mkfs.vfat -F 32 -n "ARCH_BOOT" /dev/sdb2
```

1.3.2 Назначение метки ARCH_ROOT для корневой системы BTRFS:

#bash

```
sudo mkfs.btrfs -f -L "ARCH_ROOT" /dev/sdb3
```

(Утилита isohybrid из пакета syslinux сама запишет туда MBR-загрузчик на этапе сборки ISO).

1.3.3. Проверка Показывает структуру, тип файловой системы (FSTYPE) и метку (LABEL):

#bash

```
lsblk -f
```

1.3.4 Проверка текущего монтирования
Чтобы загрузчик установился корректно, разделы должны быть смонтированы внутри вашей среды arch-chroot. Обычная структура выглядит так:

```
NAME   FSTYPE FSVER LABEL     UUID                                 FSAVAIL
FSUSE% MOUNTPOINTS
sda
├─sda1 vfat   FAT32           64D4-DC3F                             977.5M
4% /boot
└─sda2 btrfs                  53b73831-4cdb-4783-8dd1-e2342ec6c2bf   42.1G
13% /var/log
/var/cache/pacman/pkg
/home
/
sdb
├─sdb1
├─sdb2 vfat   FAT32 ARCH_BOOT AAD6-B042
└─sdb3 btrfs        ARCH_ROOT cb7b1794-2e34-456c-b4c6-b470a4d3df26
zram0  swap   1     zram0     2fad77bf-318e-45ee-8f60-87fee0c1c882
[SWAP]
```

Структура разделов на диске sdb

Для универсальной загрузки (и на старых компьютерах с BIOS, и на современных с UEFI) диск размечается в таблице GPT на три раздела:

Раздел	Назначение	Размер	Файловая система	Метка (LABEL)
/dev/sdb1	Загрузчик BIOS Boot (нужен для Syslinux/GRUB на старом железе)	1 МБ	Нет (пустой)	Нет
/dev/sdb2	Системный раздел EFI (EFI System Partition) для UEFI	1 ГБFAT32	ARCH_BOOT	
/dev/sdb3	Корневой раздел системы (RootFS)	Всё оставшееся	BTRFS	ARCH_ROOT

Монтирование корневого раздела диска в домашнюю папку.

#bash

```
sudo mount /dev/sdb3 ~/new_iso
```

Разбор команды:

- sudo: запускает команду с правами администратора (root).
- mount: команда для подключения файловой системы устройства к дереву папок.
- /dev/sdb3: третий раздел на втором жестком диске (sdb).
- ~/new_iso: целевая папка (точка монтирования) внутри вашей домашней директории.

Результат:

- содержимое раздела /dev/sdb3 станет доступно для чтения и записи внутри папки ~/new_iso.

Проверка монтирования корневого раздела.

Найти все активные монтирования, связанные с

#bash

```
mount | grep ~/new_iso
```

Разбор команды:

- mount: без параметров выводит список всех вообще примонтированных устройств в системе.
- | (конвейер): передает этот огромный список на вход следующей команде.
- grep : фильтрует строки, оставляя только те, где встречается текст .

Результат:

- вы увидите, примонтировано ли что-то в ~/new_iso или системный , а также узнаете параметры этого подключения (например, rw — чтение/запись, или ro — только чтение).

Создать папку etc внутри точки монтирования.

#bash

```
sudo mkdir -p ~/new_iso/etc
```

Разбор команды:

- mkdir: команда создания директорий.
- -p (parents): полезный флаг. Он создаст всю цепочку папок (~/new_iso/, ~/new_iso/ и etc), если какого-то из этих промежуточных каталогов еще не существует. Также этот флаг защищает от ошибок, если папка уже создана.

Результат:

- создается каталог etc. Если перед этим вы успешно выполнили первую команду (mount), то эта папка создастся прямо внутри раздела /dev/sdb3.

Проверка создания папки /etc

#bash

```
tree ~/new_iso
```

(Команда tree ~/new_iso выведет структуру каталогов и файлов внутри этой папки в виде наглядного дерева.)

1.3.6 Автоматическая генерация fstab Команда ls -ld ~/new_iso/etc/ выводит подробную информацию о самой папке etc, а не о файлах внутри неё.

#bash

```
ls -ld ~/new_iso/etc/
```

Важно: Эту команду нужно выполнять снаружи chroot-окружения (внутри установочной флешки), когда все ваши разделы уже смонтированы в папку .
Выполните команду:

#bash

```
sudo genfstab -U ~/new_iso | sudo tee -a ~/new_iso/etc/fstab
```

*(* Флаг -U указывает утилите использовать уникальные UUID разделов вместо имен вроде /dev/sdb3. Это гарантирует, что система загрузится, даже если вы вставите диск в другой ПК или другой SATA-разъем. * Команда sudo tee запускает саму утилиту записи с правами администратора (root). Это позволяет успешно сохранить сгенерированную таблицу разделов в файл fstab * Флаг -a (сокращение от append — добавить) переключает утилиту в режим добавления. Новый текст дописывается в самый конец файла, а старые данные не удаляются)*

Проверка результата

Откройте файл для проверки:

#bash

```
cat ~/new_iso/etc/fstab
```

Если вы ставили систему без подтомов (прямо в корень BTRFS), ваш файл должен выглядеть примерно так:text

```
[eva@tom1 ~]$ cat ~/new_iso/etc/fstab
# /dev/sdb3 LABEL=ARCH_ROOT
```

```
UUID=cb7b1794-2e34-456c-b4c6-b470a4d3df26 / btrfs
rw,noatime,space_cache=v2,compress=zstd,ssd 0 0
[eva@tom1 ~]$
```

> ⚠ Оптимизация для BTRFS (Рекомендуется)
> Стандартные настройки genfstab рабочие, но для BTRFS (особенно если у вас SSD-диск) крайне полезно вручную отредактировать опции монтирования для корня (/).
> Откройте файл через nano /etc/fstab и добавьте в строку с btrfs следующие параметры через запятую: ssd — включает оптимизацию под твердотельные накопители (если у вас SSD).
> compress=zstd — включает прозрачное сжатие данных (сильно экономит место и продлевает жизнь SSD).

Редактирование файла fstab

#bash

```
sudo nano ~/new_iso/etc/fstab
```

Пример оптимизированной строки:

fstab

```
# /dev/sdb3 LABEL=ARCH_ROOT
UUID=cb7b1794-2e34-456c-b4c6-b470a4d3df26   /   btrfs
rw,noatime,discard=async,space_cache=v2,compress=zstd,ssd   0 0
```

(Сохраните файл (Ctrl + O, Enter) и выйдете из nano (Ctrl + X).

Проверить, что вы всё сделали правильно

#bash

```
cat ~/new_iso/etc/fstab
```

Посчитайте количество элементов в строке с UUID (разделенных пробелами). Их должно быть ровно 6:

1. UUID=...
2. /
3. btrfs

4. rw,noatime,discard=async,space_cache=v2,compress=zstd,ssd
5. 0
6. 0

Отключение диска sdb3 от системы

Команда sudo umount -l /dev/sdb3 выполняет «ленивое» (lazy) размонтирование раздела диска.

#bash

```
sudo umount -l /dev/sdb3
```

Разбор параметров команды

- sudo: запускает операцию с правами администратора.
- umount: стандартная системная команда для отключения файловой системы.
- -l (lazy): ключевой флаг, который активирует режим «ленивого» (или отложенного) отключения.
- /dev/sdb3: целевой раздел диска, который нужно отключить.

Этап 2. Сборка RootFS внутри нового образа (~/new_iso/)

Визуальная схема структуры монтирования.

В процессе сборки (Этап 2.1) создается иерархическая структура, где один раздел накопителя-донора монтируется внутрь другого:

```
/ (Корень хоста tom_1)          <—— раздел sda2
├── boot/                          <—— загрузочный раздел хоста (/dev/sda1)
├── [BIOS Boot]                    <—— X (НЕ смонтирован / раздел /dev/sdb1)
└── home/
    └── eva/
        └── new_iso/               <—— сюда монтируется /dev/sdb3 (ARCH_ROOT)
            └── boot/              <—— сюда монтируется /dev/sdb2 (ARCH_BOOT)
```

Выполните команду для проверки текущего дерева монтирования:

#bash

```
lsblk -f
```

Как выглядит идеальный результат для вашего диска:

#bash

```
NAME   FSTYPE FSVER LABEL     UUID
FSAVAIL FSUSE% MOUNTPOINTS
sda
├─sda1 vfat   FAT32           64D4-DC3F
977.5M     4% /boot
└─sda2 btrfs                  53b73831-4cdb-4783-8dd1-e2342ec6c2bf
42.1G    13% /var/log
/var/cache/pacman/pkg
/home
/
sdb
├─sdb1
├─sdb2 vfat   FAT32 ARCH_BOOT AAD6-B042
└─sdb3 btrfs        ARCH_ROOT cb7b1794-2e34-456c-b4c6-b470a4d3df26
zram0  swap   1     zram0     2fad77bf-318e-45ee-8f60-87fee0c1c882
[SWAP]
```

Вот список из 4 ключевых вещей, которые вам нужно проверить в терминале:

1. 1. Колонка **FSTYPE** (Тип файловой системы)

Убедитесь, что система правильно видит форматирование ваших разделов:

1. Для /dev/**sdb2** (загрузочный) там должно быть строго **vfat** (или fat32).
2. Для /dev/**sdb3** (корневой) там должно быть написано **btrfs**.
3. Если там пусто, значит раздел еще не отформатирован.
4. 2. Колонка **LABEL** (Метка диска)

Проверьте, соответствуют ли имена вашему плану:

1. У загрузочного раздела должно стоять **ARCH_BOOT**.
2. У корневого раздела должно стоять **ARCH_ROOT**.
3. Метки помогают визуально не перепутать разделы местами.
4. 3. Колонка **UUID** (Уникальный идентификатор)

Это длинная строка из букв, цифр и дефисов. Именно её вы записывали в файл fstab.

1. Убедитесь, что **UUID** для /dev/**sdb3** в выводе команды символ в символ совпадает с тем **UUID**, который вы оставили в файле **~/new_iso/etc/fstab**. Если они отличаются, система выдаст ошибку при загрузке.
2. 4. Колонка **MOUNTPOINTS** (Точки монтирования)
3. В этой колонке напротив строк **sdb2** и **sdb3** должно быть абсолютно **пусто**.

2.1 Монтирование новых дисов для загрузки системы

Монтируются два раздела диска sdb в рабочую папку сборки (~/new_iso):

- **sdb1** (BIOS Boot) — **не монтируется** (нужен просто как «маяк» для старого

BIOS).
- **sdb2** (FAT32, метка ARCH_BOOT) — монтируется в папку **~/new_iso/boot**. Нужен для размещения файлов EFI-загрузчика (systemd-boot), ядра Linux (vmlinuz-linux) и загрузочного образа initramfs
- **sdb3** (Btrfs, метка ARCH_ROOT) — монтируется в корень сборки **~/new_iso**. Сюда через утилиту rsync копируется вся основная корневая файловая система (RootFS) с текущего рабочего хоста tom_1.

//

#bash

```
# mkdir -p ~/new_iso - создана в Этап1 раздел 1.1 Создание рабочих директорий
sudo mount /dev/sdb3 ~/new_iso        # Корень BTRFS монтируем в корень сборки
sudo mkdir -p ~/new_iso/boot
sudo mount /dev/sdb2 ~/new_iso/boot  # FAT32 монтируем в boot
```

1. /dev/sdb1 (BIOS Boot) — НЕ МОНТИРУЕТСЯ

Этот раздел действительно **остается несмонтированным** на протяжении всего процесса.

- В инструкции на Этапе 1.3.4 прямо указано: «Раздел sdb1 (bios boot) должен оставаться пустым и не смонтированным».
- Зачем он нужен: Этот раздел имеет крошечный размер (всего 1 МБ). Он нужен только для разметки GPT, чтобы старые материнские платы (BIOS/Legacy) могли найти загрузочный код утилиты Syslinux. Файловая система на нем не создается, файлы туда не копируются, и монтировать его в папки Linux нельзя.

3. /dev/sdb3 (Корень Btrfs) — МОНТИРУЕТСЯ

Этот раздел монтируется в первую очередь.

- В инструкции на Этапе 2.1 выполняется команда: sudo mount /dev/sdb3 ~/new_iso.
- Зачем: Это самый большой раздел, куда копируется вся операционная система (папки /etc, /usr, /var и т.д.). Папка ~/new_iso становится отображением этого диска.

2. /dev/sdb2 (папка boot) — МОНТИРУЕТСЯ

Этот раздел действительно **остается несмонтированным** на протяжении всего процесса.

- В инструкции на Этапе 1.3.4 прямо указано: «Раздел sdb1 (bios boot) должен оставаться пустым и не смонтированным».
- Зачем он нужен: Этот раздел имеет крошечный размер (всего 1 МБ). Он нужен только для разметки GPT, чтобы старые материнские платы (BIOS/Legacy) могли найти

загрузочный код утилиты Syslinux. Файловая система на нем не создается, файлы туда не копируются, и монтировать его в папки Linux нельзя.

#bash

```
lsblk -f
```

```
[eva@tom1 ~]$ lsblk -f
NAME   FSTYPE FSVER LABEL     UUID                                 FSAVAIL
FSUSE% MOUNTPOINTS
sda
├─sda1 vfat   FAT32           64D4-DC3F                             977.5M
4% /boot
└─sda2 btrfs                  53b73831-4cdb-4783-8dd1-e2342ec6c2bf   42.1G
13% /var/log
/var/cache/pacman/pkg
/home
/
sdb
├─sdb1
├─sdb2 vfat   FAT32 ARCH_BOOT AAD6-B042                              1022M
0% /home/eva/new_iso/boot
└─sdb3 btrfs        ARCH_ROOT cb7b1794-2e34-456c-b4c6-b470a4d3df26   25.5G
0% /home/eva/new_iso
zram0  swap   1     zram0     2fad77bf-318e-45ee-8f60-87fee0c1c882
[SWAP]
```

Итоговая схема п.2.1

```
[eva@tom1 ~]$ tree /home
/home
└── eva
    └── new_iso
        ├── boot
        └── etc
            └── fstab

5 directories, 1 file
```

2.2 копируем корень tom_1, исключая виртуальные и временные папки.

#bash

```
sudo rsync -aAXv --
exclude={"/dev/*","/proc/*","/sys/*","/tmp/*","/run/*","/mnt/*","/media
/*","/lost+found","/home/*/.cache/*","/home/eva/new_iso/*","/home/eva/o
riginal_iso_image/*","/etc/fstab"} / /home/eva/new_iso/
```

Проверим выполнение с вложенностью 3 уровня

Выведем только интересующие нас папку /boot и файл /etc/fstab

#bash

```
sudo find /home/eva/new_iso -maxdepth 2 \( -path "*/boot" -o -path "*/etc" -o -path "*/etc/fstab" \) | tree --fromfile
```

```
└── home
    └── eva
        └── new_iso
            ├── boot
            └── etc
                └── fstab

5 directories, 2 files
```

Проверка файла /etc/fstab

Проверим, что файл /home/eva/new_iso/etc/fstab существует, не затерт и содержит правильные данные — выведем его содержимое на экран с помощью команды cat.

#bash

```
cat /home/eva/new_iso/etc/fstab
```

```
# /dev/sdb3 LABEL=ARCH_ROOT
UUID=85f23d27-9469-4c60-89d5-79242d2e1e17       /               btrfs
rw,noatime,discard=async,space_cache=v2,compress=zstd,ssd        0 0
```

Этап 3. Настройка системы через Chroot (chroot ~/new_iso/)

Переходим внутрь создаваемой системы для изоляции настроек.

3.1 Вход в окружение

#bash

```
sudo arch-chroot ~/new_iso/
```

```
[eva@tom1 ~]$ sudo arch-chroot ~/new_iso/
[root@tom1 /]#
```

3.2 Время и Офлайн-режим

Во время установки Arch Linux с помощью скрипта archinstall или официального образа служба синхронизации времени включается автоматически. По умолчанию за это отвечает служба systemd-timesyncd

3.2.1 Настройка времени (Универсальное UTC + запрет синхронизации):

Эти команды используются для настройки системного времени и управления синхронизацией часов в Linux.
Обычно такой набор команд применяется в изолированных окружениях, контейнерах или при сборке кастомных ISO-образов, где нужно жестко зафиксировать всемирное время (UTC) и отключить фоновые сетевые службы времени.

1. Устанавливаем часовой пояс UTC / Europe/Moscow внутри chroot

#bash

```
ln -sf /usr/share/zoneinfo/UTC /etc/localtime
```

или

#bash

```
ln -sf /usr/share/zoneinfo/Europe/Moscow /etc/localtime
```

Установка системного часового пояса в UTC / Europe/Moscow.

- ln: утилита для создания ссылок между файлами.
- -s (symbolic): создает символическую ссылку (ярлык), а не жесткую копию.
- -f (force): принудительно перезаписывает старую ссылку /etc/localtime, если она уже существовала.

Как это работает:

- файл /etc/localtime указывает операционной системе, какое локальное время отображать.

Эта команда привязывает вашу систему к эталонному всемирному времени UTC (Zero timezone). Все логи и системные часы теперь будут работать без смещения на зимнее/летнее время или региональные пояса.

Проверка часового пояса (UTC / Europe/Moscow)

Выполните команду ls -l для файла /etc/localtime. Она должна показать, что этот файл является символической ссылкой, указывающей именно на файл UTC / Europe/Moscow:

#bash

```
ls -l /etc/localtime
```

Идеальный вывод:

```
# UTC
lrwxrwxrwx 1 root root 23 May 31 19:08 /etc/localtime ->
/usr/share/zoneinfo/UTC
# Europe/Moscow
lrwxrwxrwx 1 root root 33 May 31 22:35 /etc/localtime ->
/usr/share/zoneinfo/Europe/Moscow
```

Также можно запустить команду date -u — она покажет текущее системное время в формате UTC / Europe/Moscow.

2. Переносим системное время в аппаратное

#bash

```
hwclock --systohc
```

Запись текущего системного времени в аппаратные часы материнской платы.

- hwclock: утилита для работы с аппаратными часами (RTC — Real Time Clock),

которые питаются от батарейки на материнской плате вашего компьютера.

- –systohc (System to Hardware Clock): берет точное время из операционной памяти

(которое настроено в Linux) и принудительно записывает его в чип BIOS/UEFI. Зачем это нужно:

- чтобы при следующей перезагрузке компьютера материнская плата сразу передала ядру

Linux правильное время, даже если не будет интернета.

Проверка переноса времени в аппаратное

Чтобы убедиться, что время успешно записалось в аппаратную часть, просто вызовите утилиту чтения часов без флагов:

#bash

```
hwclock --show
```

Идеальный вывод:

```
2026-05-31 19:49:22.992460+00:00
```

Идеальный вывод: Команда должна вывести актуальную дату и точное время (обычно с припиской .000000+00:00), не выдавая ошибок доступа к устройству /dev/rtc.

3. Ручной "mask" службы — связываем её с /dev/null

#bash

```
ln -sf /dev/null /etc/systemd/system/systemd-timesyncd.service
```

Полная блокировка («замораживание») службы синхронизации времени.

- mask: это самая сильная форма отключения сервиса в Systemd. Она связывает файл службы с пустой директорией /dev/null.

Результат:

- службу systemd-timesyncd теперь невозможно запустить вообще никак — ни автоматически,

ни вручную, ни в качестве зависимости для других программ. Любая попытка включить её вызовет ошибку. Это гарантирует, что интернет-соединение во время работы системы случайно не изменит зафиксированное вами время.

Как это сработает на конечной машине?

Когда ваш кастомный образ загрузится на компьютере без интернета, systemd попытается прочитать конфигурацию службы systemd-timesyncd.service, наткнется на пустышку /dev/null и просто проигнорирует её запуск, вообще не тратя системные ресурсы.

Проверка ручной блокировки службы (mask)

Проверьте, куда указывает созданный ярлык службы. Для этого выведите информацию о файле сервиса:

#bash

```
ls -l /etc/systemd/system/systemd-timesyncd.service
```

Идеальный вывод:

```
lrwxrwxrwx 1 root root 9 May 31 19:44 /etc/systemd/system/systemd-
timesyncd.service -> /dev/null
```

Идеальный вывод: systemd-timesyncd.service → /dev/null

3.3 Настройка статического офлайн-репозитория

Для настройки статического офлайн-репозитория на Arch Linux скачайте нужные .pkg.tar.zst пакеты и их зависимости, поместите их в отдельную папку (например, /var/local/repo), создайте базу данных утилитой repo-add, а затем добавьте полученный путь в конфигурационный файл /etc/pacman.conf перед официальными репозиториями.

3.3.1 Создаем локальный репозиторий внутри образа из кэша tom_1, База пакетов уже скопирована вместе с /var/cache/pacman/pkg/.

Перейдем в папку /var/cache/pacman/pkg/

#bash

```
cd /var/cache/pacman/pkg/
```

вывод

```
[root@tom1 /]# cd /var/cache/pacman/pkg/
[root@tom1 pkg]#
```

Создаем локальную БД:

#bash

```
repo-add custom.db.tar.gz *.pkg.tar.zst
```

вывод

```
[root@tom1 pkg]# repo-add custom.db.tar.gz *.pkg.tar.zst

  -> Computing checksums...
  -> Creating 'desc' db entry...
  -> Creating 'files' db entry...
==> Creating updated database file 'custom.db.tar.gz'
[root@tom1 pkg]#
```

3.3.2 Настройка секции [custom] в pacman.conf для полной изоляции Чтобы система на tom_2 гарантированно не ломилась в сеть и брала пакеты только из локального кэша, мы полностью

отключаем внешние репозитории и зеркала.

3.3.3 Настраиваем pacman.conf внутри образа, чтобы он смотрел только в локальный кэш:
3.3.3.1 Откройте конфигурационный файл:

#bash

```
sudo nano /etc/pacman.conf
```

3.3.4 Настройка файла:

3.3.4.1 Найди и закомментируй (поставь # в начале строки) все стандартные репозитории: [core], [extra], [community]. Вместе с ними закомментируй строки Include = /etc/pacman.d/mirrorlist.

3.3.4.2 В самый конец файла добавь твою локальную секцию [custom]. Итоговый блок репозиториев в /etc/pacman.conf должен выглядеть строго так:

pacman.conf

```
# Полностью отключаем внешние репозитории
#[core]
#Include = /etc/pacman.d/mirrorlist

#[extra]
#Include = /etc/pacman.d/mirrorlist

# Добавляем изолированный локальный репозиторий
[custom]
SigLevel = Optional TrustAll
Server = file:///var/cache/pacman/pkg
```

После правки

```
#[core]
#Include = /etc/pacman.d/mirrorlist

#[extra-testing]
#Include = /etc/pacman.d/mirrorlist

#[extra]
#Include = /etc/pacman.d/mirrorlist

# Добавляем изолированный локальный репозиторий
[custom]
SigLevel = Optional TrustAll
Server = file:///var/cache/pacman/pkg
```

(Нажмите сочетание клавиш Ctrl + O, затем Enter для подтверждения записи файла, и наконец

Ctrl + X для выхода)

3.3.5 Проверка внутри Chroot

После сохранения файла обязательно обнови локальную базу данных, чтобы проверить работу:

#bash

```
pacman -Sy
```

вывод

```
[root@tom1 pkg]# pacman -Sy
:: Synchronizing package databases...
 custom                                  148.1 KiB   145 MiB/s 00:00
[#######################################] 100%
[root@tom1 pkg]#
```

(Результат: Система должна моментально считать базу данных custom.db напрямую из папки без единого сетевого запроса.)

3.4 Настройка сети (Статический IP флешки в ОЗУ)

Прописываем параметры для работы в ОЗУ (IP: 192.168.1.150).

Т.к. файл профиля systemd-networkd скопирован с основной системы tom_1 и имеет IP:192.168.1.72, то откроем его для редактирования:

#bash

```
nano /etc/systemd/network/20-wired.network
```

Изменим одержимое:

20-wired.network

```
[Match]
Name=en*
Name=eth*

[Network]
Address=192.168.1.150/24
Gateway=192.168.1.1
DNS=1.1.1.1
```

!!!!!!!!!!!!!!!!!!!!Я ТУТ!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!Я ТУТ!!!!!!!!!!!!!!!!!!!!!!!!!

3.5 Включение сервисов (Nginx порт 7000 + SSH)

Настройка SSH

Убедитесь, что служба OpenSSH включена, чтобы иметь возможность удаленного подключения:

#bash

```
systemctl enable sshd
```

По умолчанию Nginx слушает 80-й порт. Чтобы изменить его на 7000: Откройте конфигурационный файл в текстовом редакторе (например, nano):

3.5.1 Правим порт Nginx:

#bash

```
nano /etc/nginx/nginx.conf
```

Найдите блок server { ... } и проверьте/ измените директиву listen:

nginx.conf

```
listen       7000;
server_name  localhost;
```

Сохраните изменения (в nano: нажмите Ctrl + O, затем Enter, для выхода — Ctrl + X).
Включите службу Nginx

#bash

```
systemctl enable nginx
```

1. systemd-networkd

Это системный демон для управления сетевыми интерфейсами и подключениями

2. systemd-resolved

Это локальный кэширующий DNS-клиент (распознаватель)

Как они работают вместе:

systemd-networkd поднимает соединение и получает от провайдера IP-адрес и адреса DNS-серверов. Затем он передает эти DNS-адреса службе systemd-resolved, которая берет на себя всю обработку запросов от ваших приложений (например, браузера).

3.5.2 Активируем автозапуск:

#bash

```
systemctl enable systemd-networkd systemd-resolved
```

Вывод:

```
[root@tom1 ~]# systemctl enable systemd-networkd systemd-resolved sshd nginx
Created symlink '/etc/systemd/system/dbus-org.freedesktop.resolve1.service'
→ '/usr/lib/systemd/system/systemd-resolved.service'.
Created symlink '/etc/systemd/system/sysinit.target.wants/systemd-
resolved.service' → '/usr/lib/systemd/system/systemd-resolved.service'.
Created symlink '/etc/systemd/system/sockets.target.wants/systemd-resolved-
varlink.socket' → '/usr/lib/systemd/system/systemd-resolved-varlink.socket'.
Created symlink '/etc/systemd/system/sockets.target.wants/systemd-resolved-
monitor.socket' → '/usr/lib/systemd/system/systemd-resolved-monitor.socket'.
[root@tom1 ~]#
```

Проверка автозапуска служб

В среде chroot команда systemctl status может работать некорректно из-за отсутствия запущенного менеджера инициализации (systemd). Чтобы гарантировать, что службы добавлены в автозагрузку, выполните:

#bash

```
systemctl is-enabled sshd
systemctl is-enabled nginx
systemctl is-enabled systemd-networkd
systemctl is-enabled systemd-resolved
```

Если службы добавлены в автозагрузку, все команды вернут ответ enabled.

Вывод:

```
[root@tom1 ~]# systemctl is-enabled sshd
enabled
[root@tom1 ~]# systemctl is-enabled nginx
enabled
[root@tom1 ~]# systemctl is-enabled systemd-networkd
enabled
[root@tom1 ~]# systemctl is-enabled systemd-resolved
enabled
[root@tom1 ~]#
```

3.6 Настройка Swap в ОЗУ (zram-generator)

#bash

```
nano /etc/systemd/zram-generator.conf
```

Содержимое:

zram-generator.conf

```
[zram0]
zram-size = ram / 2
compression-algorithm = zstd
```

3.7 Перезапись fstab под новые метки (LABEL)

#bash

```
# Так как rsync скопировал реальный fstab с UUID хоста tom_1,
# мы полностью очищаем его внутри chroot, чтобы Live-система на tom_2
работала в ОЗУ без привязки к дискам.
true > /etc/fstab
```

(Если система при загрузке в ОЗУ затребует UUID дисков, то пробуем этот пункт 3.7 Перезапись fstab под новые метки (LABEL):

#bash

```
cat <<EOF > /etc/fstab
LABEL=ARCH_ROOT    /        btrfs    defaults,noatime,compress=zstd
0 0
LABEL=ARCH_BOOT    /boot  vfat  defaults,fmask=0077,dmask=0077  0  2
EOF
```

)

nano /etc/fstab

3.8 Установка и обновление EFI-загрузчика

Перемонтируйте раздел повторно Теперь примените новые параметры монтирования из обновленного fstab:

mount -o remount /boot

Смонтируйте вручную с правильными правами

mount -t vfat -o fmask=0077,dmask=0077 /dev/disk/by-label/ARCH_BOOT /boot

#bash

```
bootctl install
```

Выход из chroot:

#bash

```
exit
```

Этап 4. Конфигурация загрузчика systemd-boot и Syslinux

Установка загрузчика (Выполняется на хосте tom_1. На этом этапе разделы диска sdb все еще примонтированы в ~/new_iso/!):

4.1 Настройка параметров загрузки (loader.conf)

#bash

```
sudo nano ~/new_iso/boot/loader/loader.conf
```

Содержимое (выбор 3 секунды):

loader.conf

```
inidefault arch
timeout 3
console-mode max
```

```
editor no
```

4.2 Настройка записи загрузки параметры ядра (arch.conf)

Привязка к глобальной метке ARCH_ROOT, COM-порт и отключение прерываний Hyper-V:

#bash

```
sudo nano ~/new_iso/boot/loader/entries/arch.conf
```

Содержимое:

ini

```
# Для ~/new_iso/boot/loader/entries/arch.conf и isolinux.cfg
title Arch Linux Custom Live (SquashFS Manual)
linux /vmlinuz-linux
initrd /initramfs-linux.img
options archisolabel==ARCH_202605
archisobasedir=/arch/x86_64/airootfs.sfs rw console=ttyS0,115200n8
earlyprintk=ttyS0,115200 hv_utils.disable_gpadl_match=1
```

4.3 Подготовка Syslinux (BIOS) для физического железа

#bash

```
sudo pacman -S syslinux --noconfirm
sudo mkdir -p ~/new_iso/boot/syslinux
sudo cp
/usr/lib/syslinux/bios/{isolinux.bin,ldlinux.c32,libcom.c32,libutil.c32
,vesamenu.c32} ~/new_iso/boot/syslinux/
```

#bash

```
sudo nano ~/new_iso/boot/syslinux/isolinux.cfg
```

Содержимое:

ini

```
UI vesamenu.c32
PROMPT 0
```

```
TIMEOUT 30
DEFAULT arch

LABEL arch
LINUX /vmlinuz-linux
INITRD /initramfs-linux.img
APPEND archisolabel==ARCH_202605
archisobasedir=/arch/x86_64/airootfs.sfs rw console=ttyS0,115200n8
earlyprintk=ttyS0,115200 hv_utils.disable_gpadl_match=1
```

4.4. Настройка родного initramfs для поддержки SquashFS и OverlayFS

Чтобы система загрузилась в режиме «только чтение» из SquashFS, но позволяла изменять файлы в ОЗУ (принцип LiveCD), стандартный initramfs должен уметь работать с модулями loop и overlay.

Выполните следующие действия на хосте tom_1: Зайдите в окружение Chroot:

#bash

```
sudo arch-chroot ~/new_iso/
```

Включите поддержку модулей в конфигурации сборщика:

#bash

```
nano /etc/mkinitcpio.conf
```

Найдите строку MODULES=(...) и добавьте туда модули для работы с петлевыми устройствами и сжатыми файловыми системами:

#bash

```
MODULES=(loop overlay squashfs vfat btrfs)
```

Проверьте хуки: В строке HOOKS=(...) убедитесь, что присутствуют systemd и block:

#bash

```
HOOKS=(base udev modconf memdisk archiso_loop_mnt archiso
archiso_pxe_common archiso_pxe_nbd archiso_pxe_http archiso_pxe_nfs
block filesystems keyboard)
```

Пересоберите initramfs: Генерируем новый образ загрузки, который теперь аппаратно готов

смонтировать наш будущий .sfs файл:

#bash

```
mkinitcpio -p linux
```

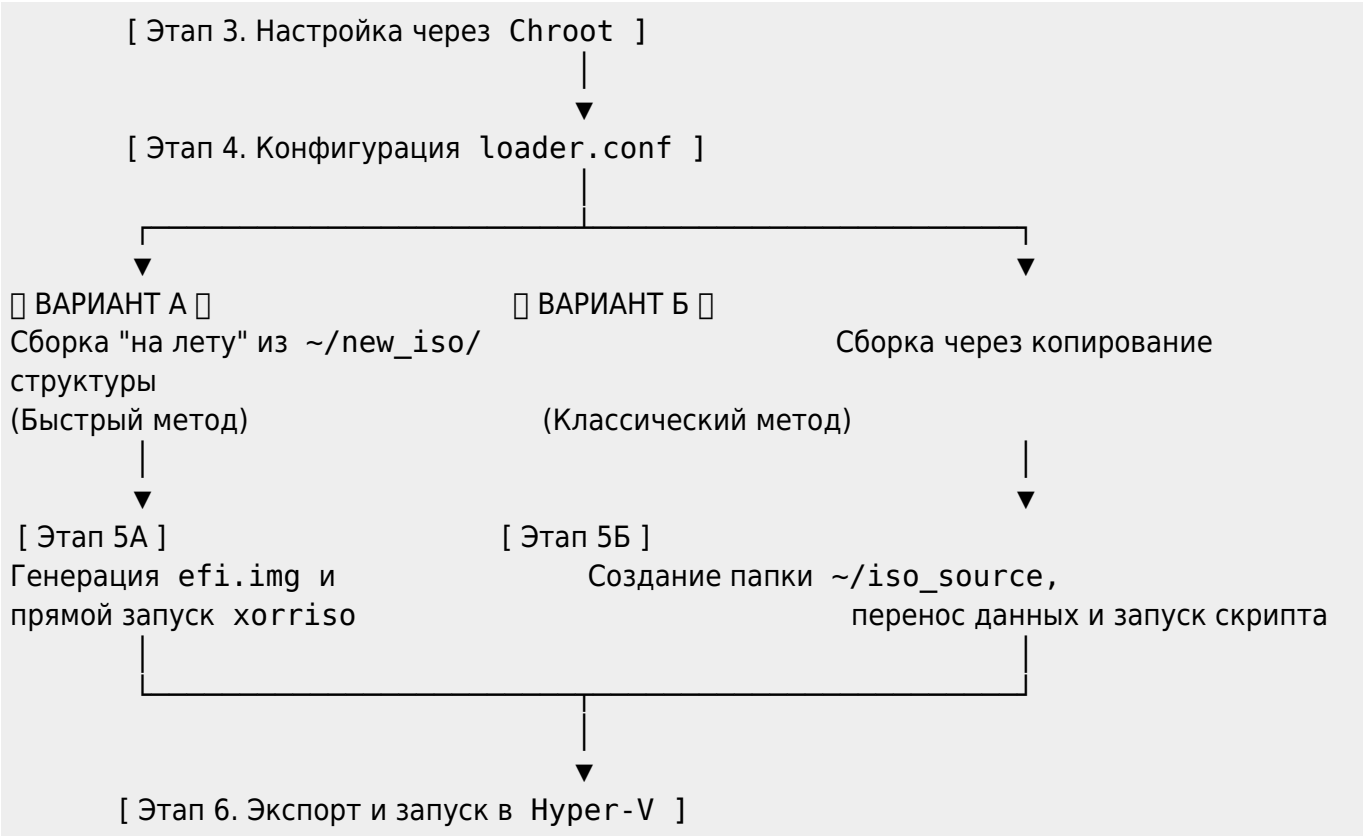
Выйдите из chroot:

#bash

```
exit
```

Этап 5. Выбор варианта Генерация UEFI-образа, SquashFS и сборка ISO для через xorriso

 ВАРИАНТ А
Сборка «на лету» прямо из ~/new_iso/ (Быстрый метод)
Метод собирает ISO прямо из рабочей директории, где смонтирован диск sdb.
(Идеально для быстрых тестов.)
 ВАРИАНТ Б
Сборка через изолированную структуру ~/iso_source (Классический метод)



Этап 5A. Генерация UEFI-образа, SquashFS и сборка ISO для через xorriso

 ВАРИАНТ А
Сборка «на лету» прямо из ~/new_iso/ (Быстрый метод)
Метод собирает ISO прямо из рабочей директории, где смонтирован диск sdb.
(Идеально для быстрых тестов.)

5A.1 Создание внутренней структуры Arch ISO и сжатие RootFS

1. Создаем целевой каталог внутри сборки

#bash

```
mkdir -p ~/new_iso/arch/x86_64/
```

2. Упаковываем RootFS во временный файл в /tmp/ хоста

#bash

```
sudo mksquashfs ~/new_iso /tmp/airootfs.sfs -comp zstd -noappend -e
arch tmp lost+found
```

3. Перемещаем готовый изолированный образ внутрь структуры

#bash

```
sudo mv /tmp/airootfs.sfs ~/new_iso/arch/x86_64/airootfs.sfs
```

5A.2 Создание внутреннего efi.img внутри структуры

1. Создаем образ efi.img во временной папке хоста в ~/new_iso/boot

#bash

```
sudo dd if=/dev/zero of=~/new_iso/boot/efi.img bs=1M count=64
status=none
sudo mkfs.vfat -F 16 -n "ARCH_202605" ~/new_iso/boot/efi.img >
/dev/null
```

2. Монтируем его для наполнения

#bash

```
mkdir -p /tmp/efi_mnt
sudo mount -o loop /tmp/efi.img /tmp/efi_mnt
```

3. Копируем файлы загрузчика ИЗ смонтированного диска во временный efi.img

#bash

```
sudo mkdir -p /tmp/efi_mnt/EFI/BOOT /tmp/efi_mnt/loader/entries
sudo cp ~/new_iso/boot/EFI/BOOT/BOOTX64.EFI /tmp/efi_mnt/EFI/BOOT/
sudo cp ~/new_iso/boot/loader/loader.conf /tmp/efi_mnt/loader/
sudo cp ~/new_iso/boot/loader/entries/arch.conf
/tmp/efi_mnt/loader/entries/
```

4. Размонтируем и удаляем точку монтирования

#bash

```
sudo umount /tmp/efi_mnt
rmdir /tmp/efi_mnt
```

5A.3 Прямой запуск xorriso сборки гибридного ISO

Запускаем сборку из директории ~/new_iso/. Результат положим в корень домашней директории.

#bash

```
sudo xorriso -as mkisofs \
  -iso-level 3 \
  -full-iso9660-filenames \
  -volid "ARCH_202605" \
  -eltorito-boot boot/syslinux/isolinux.bin \
  -eltorito-catalog boot/syslinux/boot.cat \
  -no-emul-boot -boot-load-size 4 -boot-info-table \
  -isohybrid-mbr /usr/lib/syslinux/bios/isohdpfx.bin \
  -eltorito-alt-boot \
  -e boot/efi.img \
  -no-emul-boot -isohybrid-gpt-basdat \
  -hide EFI \
  -hide loader \
  -output ~/ARCH_202605.iso \
  ~/new_iso/
```

5А.4 Безопасное размонтирование диска донора

Только теперь, когда ISO-образ успешно создан, освобождаем накопитель:

#bash

```
# Размонтируем разделы диска sdb, которые были подключены к хосту tom_1
sudo umount ~/new_iso/boot
sudo umount ~/new_iso
rm -f /tmp/efi.img
```

> Примечание: «Этап сборки завершен. Пропустите Вариант Б и переходите сразу к Этапу 6».

Этап 5Б. Подготовка структуры и сборка ISO через xorriso

 ВАРИАНТ Б Сборка через изолированную структуру ~/iso_source (Классический метод)

> Этот этап применяется, если не использовался Вариант А, позволяя упаковать систему в SquashFS и создать гибридный ISO-образ, используя временную директорию ~/iso_source.

(Метод копирует все данные в отдельную директорию на хосте, освобождая диск донор sdb сразу.)

5Б.1 Создание структуры и копирование данных

Создаем рабочую директорию и синхронизируем в нее данные, исключая временный образ EFI:
Создаем чистую директорию

#bash

```
mkdir -p ~/iso_source
```

Синхронизируем содержимое sdb в папку сборки для xorriso

#bash

```
sudo rsync -aAXv --exclude={"/boot/efi.img"} ~/new_iso/ ~/iso_source/
```

5Б.2 Освобождение диска-донора

Размонтируем исходный диск sdb, так как данные уже скопированы в iso_source:

#bash

```
sudo umount ~/new_iso/boot
sudo umount ~/new_iso
```

5Б.3 Упаковка RootFS в SquashFS

Упаковываем файловую систему в сжатый образ airootfs.sfs с использованием zstd, исключая служебные каталоги:

1. Создаем целевой каталог внутри изолированной структуры

#bash

```
mkdir -p ~/iso_source/arch/x86_64/
```

2. Упаковываем RootFS во временный файл из папки ~/iso_source, исключая arch, tmp и lost+found

#bash

```
sudo mksquashfs ~/iso_source /tmp/airootfs.sfs -comp zstd -noappend -e
arch tmp lost+found
```

3. Перемещаем готовый изолированный образ внутрь правильной структуры

#bash

```
sudo mv /tmp/airootfs.sfs ~/iso_source/arch/x86_64/airootfs.sfs
```

5Б.4 Создание скрипта сборки ISO глобальной меткой ARCH_202605.

Так как мы собираем чистый гибридный UEFI/BIOS образ вручную (без archiso), нам нужен готовый скрипт, который соберет структуру папки ~/iso_source/ в правильный .iso файл. Выполняется на хосте tom_1, вне Chroot.

5Б.4.1. Создай файл скрипта

#bash

```
nano ~/build_iso.sh
```

5Б.4.2. Вставь в него следующий код

#bash

```
#!/bin/bash

# Настройка путей
SOURCE_DIR="$HOME/iso_source"
OUTPUT_ISO="$HOME/ARCH_202605.iso"
VOLUME_ID="ARCH_202605"
TMP_EFI_MNT="/tmp/efi_mnt"

echo "=== Старт автоматической сборки гибридного ISO (BIOS + UEFI) ==="

# 1. Проверка утилит
if ! command -v xorriso &> /dev/null; then
    echo "Ошибка: xorriso не установлен. Выполните: sudo pacman -S
xorriso"
    exit 1
fi

# 2. Безопасный бэкап файлов загрузчика перед операциями, если sdb2 еще
примонтирован
# Если sdb2 уже размонтирован, скрипт возьмет файлы из структуры копии
echo "→ Синхронизация файлов загрузчика..."
mkdir -p /tmp/loader_backup/EFI/BOOT
mkdir -p /tmp/loader_backup/loader/entries

if mountpoint -q ~/new_iso/boot; then
    cp -r ~/new_iso/boot/EFI /tmp/loader_backup/
    cp -r ~/new_iso/boot/loader /tmp/loader_backup/
else
    if [ -d "$SOURCE_DIR/boot/EFI" ]; then
        cp -r "$SOURCE_DIR/boot/EFI" /tmp/loader_backup/
        cp -r "$SOURCE_DIR/boot/loader" /tmp/loader_backup/
    else
        echo "Критическая ошибка: Файлы UEFI-загрузчика не найдены ни в
~/new_iso/boot, ни в $SOURCE_DIR/boot!"
        exit 1
    fi
fi
```

```
# 3. Генерация EFI-образа для загрузки UEFI
echo "→ Подготовка EFI boot image..."
rm -f "$SOURCE_DIR/boot/efi.img"
dd if=/dev/zero of="$SOURCE_DIR/boot/efi.img" bs=1M count=64
status=none
mkfs.vfat -F 16 -n "ARCH_202605" "$SOURCE_DIR/boot/efi.img" > /dev/null

# 4. Монтируем efi.img и копируем туда файлы загрузчика systemd-boot
mkdir -p /tmp/efi_mnt
sudo mount -o loop "$SOURCE_DIR/boot/efi.img" $TMP_EFI_MNT


# 5. Проверяем успешность монтирования перед тем, как работать с директорией
if mountpoint -q $TMP_EFI_MNT; then
    echo "→ Наполнение efi.img файлами загрузчика..."
    sudo mkdir -p $TMP_EFI_MNT/EFI/BOOT
    sudo mkdir -p $TMP_EFI_MNT/loader/entries

    # Копируем из гарантированного бэкапа
    sudo cp /tmp/loader_backup/EFI/BOOT/BOOTX64.EFI
$TMP_EFI_MNT/EFI/BOOT/
    sudo cp /tmp/loader_backup/loader/loader.conf $TMP_EFI_MNT/loader/
    sudo cp /tmp/loader_backup/loader/entries/arch.conf
$TMP_EFI_MNT/loader/entries/

    sudo umount $TMP_EFI_MNT
    rmdir $TMP_EFI_MNT
    rm -rf /tmp/loader_backup
else
    echo "Критическая ошибка: Не удалось примонтировать efi.img!"
    rmdir $TMP_EFI_MNT
    exit 1
fi

# 6. Безопасная проверка готовности каталога boot перед сборкой ISO
# Вместо деструктивного удаления файлов, мы просто проверяем наличие efi.img
echo "→ Проверка загрузочной структуры в boot..."
if [ ! -f "$SOURCE_DIR/boot/efi.img" ]; then
    echo "Критическая ошибка: efi.img отсутствует в $SOURCE_DIR/boot/!"
    exit 1
fi

# Чтобы xorriso не затягивал дублирующие папки EFI и loader в корень ISO
# (они уже упакованы внутрь efi.img), мы укажем утилите xorriso исключить их
# прямо во время сборки на Шаге 7 с помощью флага -hide.

# 7. Сборка полноценного гибридного ISO через xorriso
echo "→ Запуск xorriso (Сборка гибридного образа)..."
xorriso -as mkisofs \
  -iso-level 3 \
```

```
  -full-iso9660-filenames \
  -volid "$VOLUME_ID" \
  -eltorito-boot boot/syslinux/isolinux.bin \
  -eltorito-catalog boot/syslinux/boot.cat \
  -no-emul-boot -boot-load-size 4 -boot-info-table \
  -isohybrid-mbr /usr/lib/syslinux/bios/isohdpfx.bin \
  -eltorito-alt-boot \
  -e boot/efi.img \
  -no-emul-boot -isohybrid-gpt-basdat \
  -hide EFI \
  -hide loader \
  -output "$OUTPUT_ISO" \
  "$SOURCE_DIR/"


if [ $? -eq 0 ]; then
    echo "=== Сборка успешно завершена! ==="
    echo "Файл образа: $OUTPUT_ISO"
    echo "Этот образ готов к записи через Rufus (в режиме DD/ISO) для флешек ИЛИ прямого монтирования в Hyper-V Gen1/Gen2."
else
    echo "=== Ошибка при сборке ISO ==="
    exit 1
fi
```

5Б.5 Запуск сборки

Делаем скрипт исполняемым и запускаем его для создания финального образа:

#bash

```
chmod +x ~/build_iso.sh
~/build_iso.sh
```

(Примечание: Параметры xorriso могут адаптироваться под структуру папки boot вашего диска sdb).

Скрипт автоматически упакует систему, создаст правильный UEFI-загрузочный сектор efi.img с твоей глобальной меткой ARCH_202605 и положит готовый файл в твою домашнюю директорию.

Этап 6. Экспорт ISO в Windows и монтирование в CD-привод Hyper-V (tom_2)

6.1 Передача файла на Windows-хост

Используте WinSCP на Windows:

cmd

```
pscp eva@192.168.1.72:/home/eva/ARCH_202605.iso C:\ISO\
```

6.2 Запись на arch-flash (опционально, для физ. теста)

Открываешь Rufus, выбираешь флешку, выбираешь созданный ARCH_202605.iso и пишешь в режиме DD/ISO.

6.3 Монтирование нового ISO-образа в CD-привод Hyper-V (для tom_2)

- Открой Диспетчер Hyper-V.
- Выбери изолированную виртуальную машину tom_2.
- Нажми Параметры (Settings) → vIDE-контроллер или SCSI-контроллер (в зависимости от поколения ВМ Gen1/Gen2).
- Выбери Накопитель DVD (DVD Drive).
- Установи переключатель в положение Файл образа (ISO) (Image file).
- Нажми Обзор (Browse) и укажи путь к скачанному файлу C:\ISO\ARCH_202605.iso.
- Нажми Применить (Apply) и запусти ВМ tom_2.

From:
https://wwoss.ru/ - **worldwide open-source software**

Permanent link:
https://wwoss.ru/doku.php?id=30.05.2026

Last update: **2026/06/03 07:31**