Сборка Headless Arch ISO с WebUI-инсталлятором и управлением

1. Вводные данные и Архитектура Стенда

Термины и определения

- **Ubuntu Server** — это специализированная операционная система с открытым исходным кодом от компании Canonical, разработанная для работы на серверах, в облаках и центрах обработки данных. В отличие от версии для ПК, по умолчанию она не имеет графического интерфейса и полностью управляется через командную строку.
- **Arch (Arch Linux)** — это популярный, легковесный дистрибутив Linux, созданный для опытных пользователей. Он предлагает «чистую» систему без предустановленного лишнего ПО, предоставляя полный контроль над настройками, и использует модель обновлений rolling release (постоянные обновления вместо редких глобальных версий).
- **Headless Arch ISO** — это модифицированный образ установочного диска Arch Linux, который автоматически запускает SSH-сервер и разрешает удаленное подключение без физического монитора и клавиатуры. Это позволяет устанавливать ОС на сервер или виртуальную машину полностью по сети с другого компьютера.
- **Hyper-V** — это встроенная система аппаратной виртуализации от Microsoft. Она позволяет создавать изолированные виртуальные машины (ВМ) на одном компьютере.
- **ISO-образ** (файл с расширением .iso) — это цифровой архив, представляющий собой точную, побайтовую копию («слепок») диска (CD, DVD или Blu-Ray).
- **UUID** (Universally Unique Identifier) — это универсальный 128-битный уникальный идентификатор, который файловая система автоматически присваивает диску или разделу при форматировании.
- **LABEL** (global volume label) — это понятное человеку имя, присвоенное разделу жесткого диска или логическому тому.
- **Sudo** (Superuser Do) — это утилита, позволяющая обычному пользователю выполнять команды с правами администратора (root).
- **Pacman** (Package Manager) — это официальный консольный менеджер пакетов, разработанный специально для Arch Linux. Он выполняет роль главного «инструментария» для установки, удаления, обновления и отслеживания программ в системе.
- **sda** / **sdb** (SCSI Disk) — это системные имена для ваших физических накопителей,где a, b, c — латинские буквы, обозначающие порядок, в котором система обнаружила диски.

1.1. Назначение Headless Arch ISO-images

цели создания автономного Headless-образа для серверов «вслепую» without Internet. реальный headless-сервер - без интернета и без монитора.(supermicro, 2 none-name (старый и новый) и старый HP)

Наша цель:

собрать новый, полностью универсальный ISO-образ, который будет находить флешку не по жесткому UUID, а по глобальной метке тома (LABEL=ARCH_202605).

1.2. Конфигурация и минимальные требования к оборудованию

1.2.1. Рабочая станция (DeskTop)

CPU	AMD Ryzen 7 3700X 8-Core Processor (3.59GHz)
RAM DDR4 48.0 GB	AMD Radeon R7 2x8GB 2666 MHz (R748G2606U2S-U) AMD Radeon R7 2x16GB 2666 MHz (R7416G2606U2S-U)
Motherboard	GIGABYTE B550 AORUS ELITE V2
Solid state drive	Gigabyte Aorus Gen4 5000E M.2 5000/3800MB/s(R/W) (AG450E500G-G) Gigabyte Gen4 4000E M.2 3600/3000MB/s(R/W) (G440E500G)
Flash drive	Smartbuy Crown 8 ГБ USB 3.1 (SB8GBCRW-BL) Smartbuy Crown 16 ГБ USB 2.0 (SB16GBCRW-W)
Operating system	Microsoft Windows 10 Pro 1909 (19H2) Build 18363.476 x64
Virtual machine	Microsoft Hyper-V Manager 10.0.18362.1]
Remote ssh client	PuTTY 64bit 0.84 not portable
Dev environment	Arch linux x86_64 2026.05.01
Code editor	Notepad++ v8.9.6.1
Web browser	Google Chrome 148.0.7778.179 64 bit / Mozilla Firefox 151.0.2 64-bit / Opera One 131.0.5877.74
Wiki system	Dokuwiki Librarian 2025-05-14b

1.2.3. Рекомендуемая рабочая станция

Конфигурация вашего оборудавания будет соответственно отличаться, но это так-же будет работать, заняв больше времени при установке и рендере карт на устройстве с минимальными параметрами:

- Motherboard > virtualization support
- CPU > 4 ядер
- RAM > 4Gb
- HDD > 300Mb
- OS > Windows 10 (Pro) с Hyper-V

1.2.4. Тестовые сервера

В данном руководстве установка программного обеспечения производиться на оборудование следующей конфигурации:

CPU	AMD EPYC™ 7551P OEM
Motherboard	Supermicro MBD-H11SSL-I
Solid state drive	Raid5 1tb M.2 Samsung 970 EVO Plus

RAM 512.0 GB	DDR4 3200MHz DIMM ECC Reg Micron

1.2.5. Рекомендуемая сервер

Конфигурация вашего оборудавания будет соответственно отличаться, но это так-же будет работать, заняв больше времени при установке и рендере карт на устройстве с минимальными параметрами:

- Motherboard > virtualization support
- CPU > 2 ядер (x86_64 VT-x/AMD-V)
- RAM > 4Gb
- HDD > 30Gb

1.3. Схема распределения тестовой среды

1.3.1. Распределение ролей при работе в Hyper-V (Virtual Machine)

- **tom_1** - эталонный хост, основная тестовая ВМ в Hyper-V для сборки iso образа, с доступом в интернет, с рабочим arch linux, на котором разверачивается Nginx (порт 7000) и работает SSH для PuTTY, созданы пользователи root и eva и им заданы пороли.
- **arch-flash** - виртуальный диск ы Hyper-V (носитель с записанным iso образом), куда мы через rufus записываем, созданный нами ISO-образ на tom_1
- **tom_2** - изолированный виртуальный хост, ВМ в Hyper-V без доступа в интернет, куда мы подключим arch-flash и развернем arch linux, на котором развернут Nginx (порт 7000) и работает SSH для PuTTY, созданы пользователи root и eva и им заданы пороли.

- Добавление ролей при работе с физическим железом
  - **usb-arch-server** - физический usb носитель
  - **arch-server** - физический сервер (supermicro/hp/no-name)

1.3.2. Администраторы сервера

- root - по умолчанию в Arch linux
- eva - из установленного Arch linux
- admin - новый создаваемый администратор (замена eva)

2. Выравнивание версий и подготовка окружения на tom_1

Чтобы предотвратить конфликт драйверов и панику модуля ZRAM на ранних секундах загрузки флешки, ядро загрузчика и модули внутри SquashFS-слепка должны совпадать символ в символ.

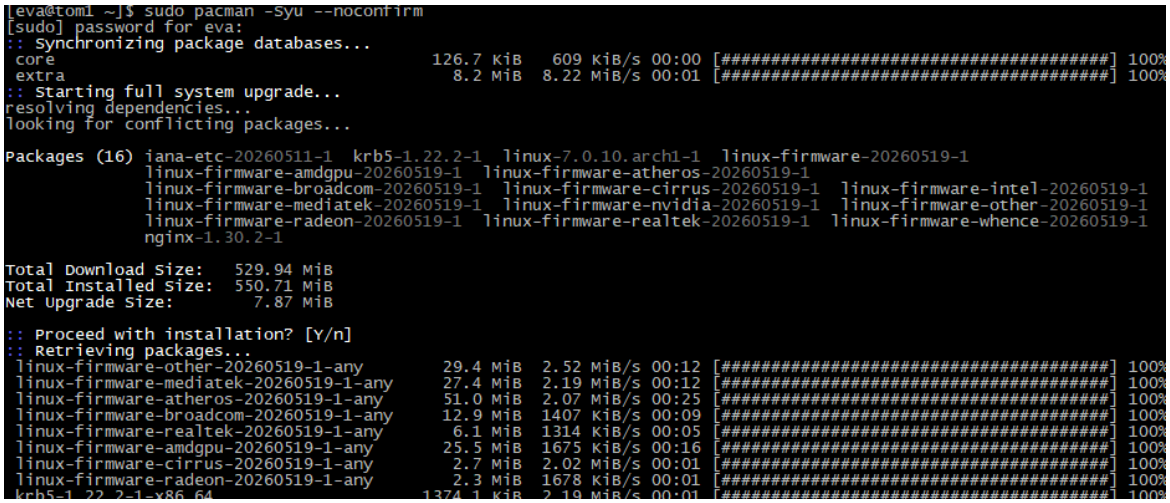
2.1. Тотальное обновление и фиксация ядра хоста

2.1.1. Обновление ядра хоста

Зайдите на чистый tom_1 по SSH под пользователем eva и обновите индекс пакетов и само ядро хоста:

#bash

```
sudo pacman -Syu --noconfirm
```



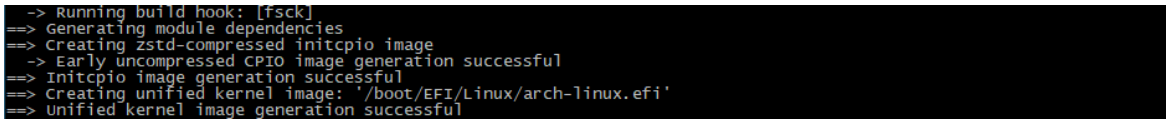
2.1.2. Фиксация эталонной версии ядра

Посмотрите на строчку генерации образа виртуального диска:

- *==⇒ Starting build: '7.0.10-arch1-1'*

Наша новая зафиксированная версия ядра: 7.0.10-arch1-1.
Обратите внимание, как утилита mkinitcpio автоматически пересобрала не просто обычный виртуальный диск, а создала единый объединенный образ:

- **Creating unified kernel image: '/boot/EFI/Linux/arch-linux.efi'**



Это означает, что на tom_1 теперь развернута абсолютно чистая, монолитная, современная база ядра. Конфликт версий официально предотвращен еще на этапе фундамента.
Поскольку в процессе этого тотального обновления pacman физически заменил старые файлы ядра в каталоге /boot/ на новые, текущее запущенное в оперативной памяти tom_1 ядро всё

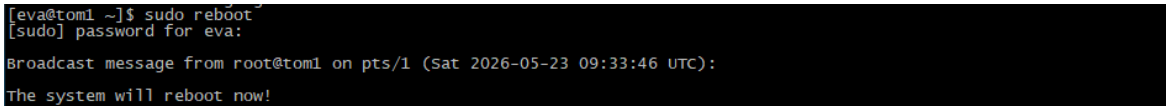
еще имеет старый индекс, а на диске уже лежат файлы версии 7.0.9-arch2-1.

2.1.2.1. Перезагрузка системы

Чтобы система tom_1 полностью приняла новое ядро и зафиксировала его в оперативной памяти, нам нужно отправить виртуалку в чистую перезагрузку.
Прямо в консоли PuTTY выполните команду:

#bash

```
sudo reboot
```



Подождите около 20–30 секунд, пока ВМ tom_1 сделает круг перезапуска в Hyper-V.
Подключитесь к tom_1 заново через PuTTY под пользователем eva

2.1.3. Загрузка и проверка

Как только зайдете обратно, выполните команду сверки

#bash

```
uname -r
```

Убедимся, что в ОЗУ хоста теперь честно светится 7.0.10-arch1-1, и только после этого медленно и аккуратно двинемся дальше по нашему новому руководству.

3. Проверка и установка ПО

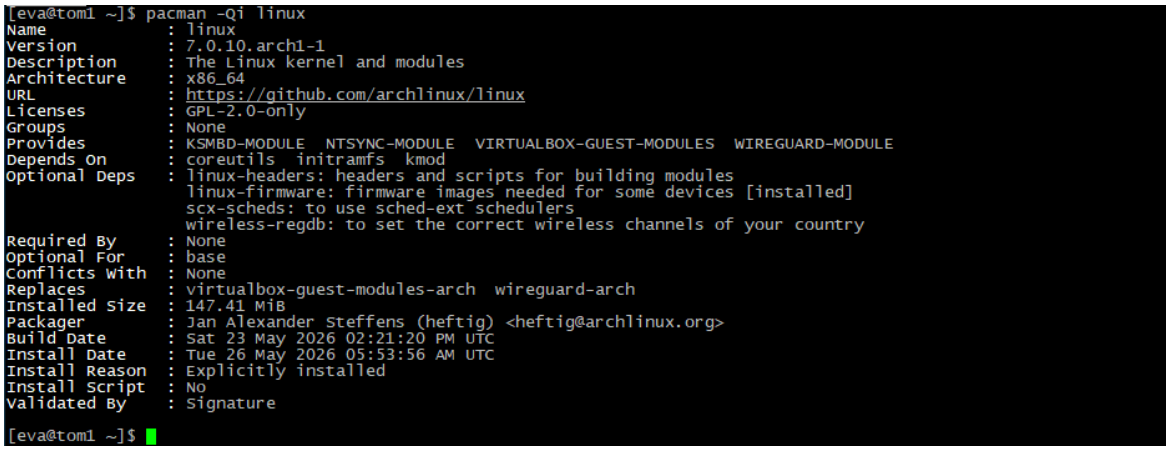
3.1. Проверка ПО

- linux: ядро операционной системы.
- linux-firmware: набор микропрограмм (драйверов) для компьютерного железа.
- https://wiki.archlinux.org/title/Systemd-boot/systemd-boot: простой и быстрый загрузчик, встроенный в системную среду systemd
- grub: популярный универсальный загрузчик

3.1.1. Проверка пакета linux

#bash

```
pacman -Qi linux
```



(В выводе отобразилась подробная информация об уже установленном в системе пакете ядра linux, его версию (7.0.10.arch1-1), архитектуру (x86_64) и т.д.)

3.1.2. Проверка пакета linux-firmware

#bash

```
pacman -Qi linux-firmware
```

(В выводе отобразилась подробная информация об уже установленном пакете прошивок linux-firmware, в частности информацию о файлах встроенного программного обеспечения

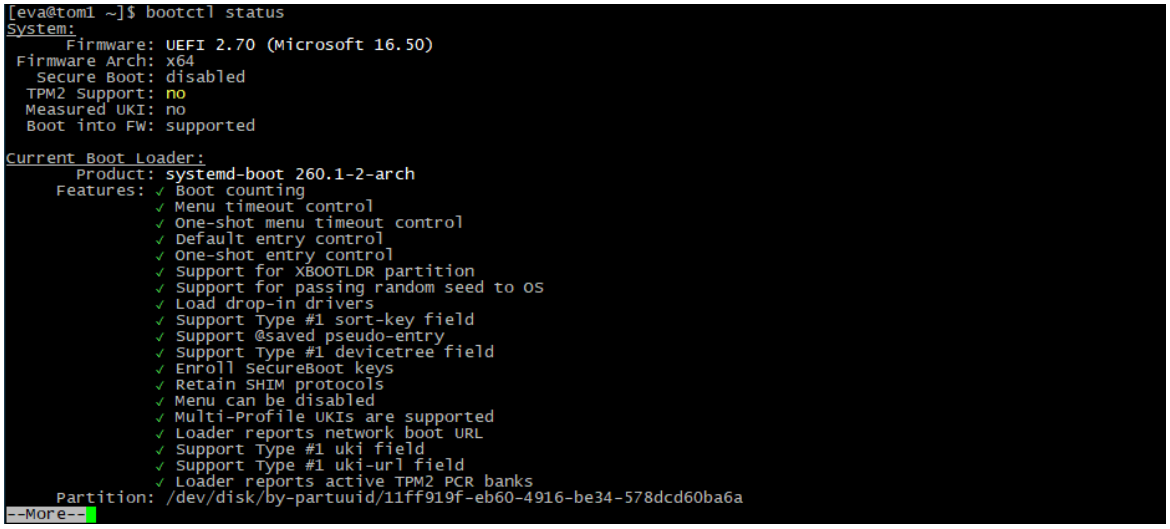
(микрокод/прошивки) для работы различного оборудования — например, Wi-Fi адаптеров, видеокарт, звуковых плат и Bluetooth-модулей.)

3.1.3. Проверка загрузчика

Мы можем проверить это за с помощью утилиты bootctl

#bash

```
bootctl status
```



- *Если вы увидите детальную информацию с текстом вроде «Features: ✓ boot-info» и версиями в строках «Product: systemd-boot», значит, ваша система использует systemd-boot.*
- *Если команда выдаст ошибку или укажет, что система загружена не в режиме UEFI (например, «System is not booted with EFI»), то у вас используется классический BIOS/MBR загрузчик, либо другой менеджер загрузки.*

(Для выхода CTRL+C)

Проверка пакета grub в /var/cache/pacman/pkg/

Проверить наличие кэшированных файлов пакета grub можно с помощью стандартных команд терминала для поиска файлов. pacman хранит в этом каталоге скачанные архивные файлы с расширением .pkg.tar.zst (или .pkg.tar.xz).

#bash

```
ls /var/cache/pacman/pkg/ | grep grub
```

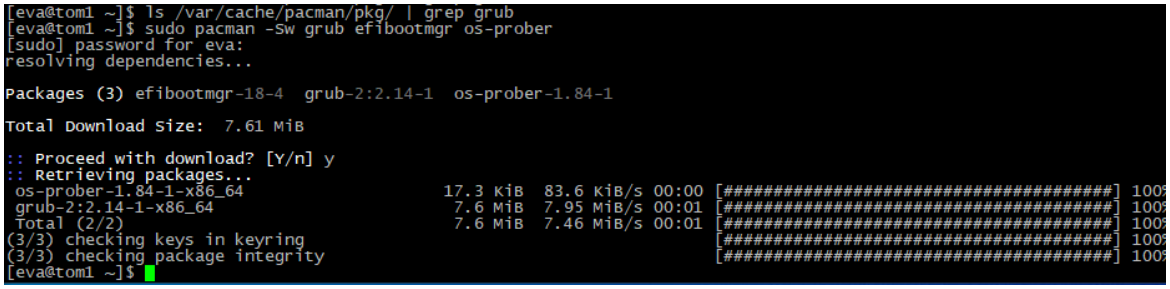
(выведет список всех файлов в кэше, в имени которых содержится слово «grub». Если пакет там есть, вы увидите файлы вида grub-2.12-1-x86_64.pkg.tar.zst, иначе пустой вывод)

Скачивание пакета в кэш

Для загрузки пакета без установки используйте ключ -Sw. Также сразу скачать сопутствующие пакеты, которые понадобятся для настройки GRUB:

#bash

```
sudo pacman -Sw grub efibootmgr os-prober
```

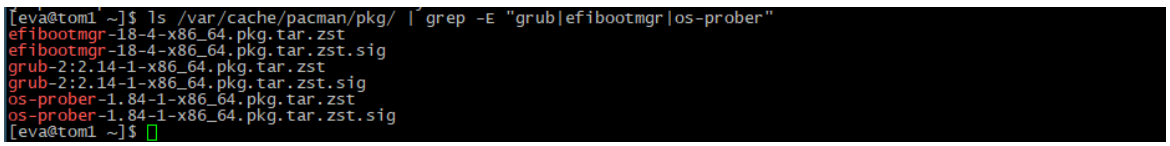


Проверка загруженного пакета grub с зависимостями

Чтобы проверить наличие всех трех пакетов в кэше одной командой, используйте расширенный grep (grep -E) и вертикальную черту | в качестве разделителя «ИЛИ»

#bash

```
ls /var/cache/pacman/pkg/ | grep -E "grub|efibootmgr|os-prober"
```



4. Установка необходимых пакетов

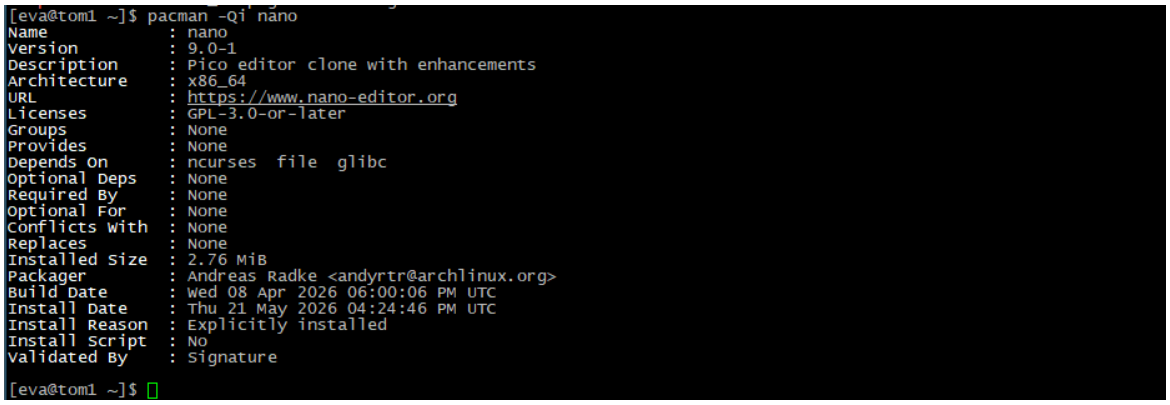
4.1. Пакет редактора nano

Пакет nano в Arch Linux — это официальный пакет, который содержит одноимённый консольный текстовый редактор nano

4.1.1. Проверка пакета nano в системе

#bash

```
pacman -Qi nano
```



(В выводе отобразилась подробная информация об уже установленном пакете редактора nano.)

4.1.2. Проверка пакета nano в /var/cache/pacman/pkg/

Проверить наличие кэшированных файлов пакета nano можно с помощью стандартных команд терминала для поиска файлов. pacman хранит в этом каталоге скачанные архивные файлы с расширением .pkg.tar.zst (или .pkg.tar.xz).

#bash

```
ls /var/cache/pacman/pkg/ | grep nano
```

(Этот вывод означает, что в кэше вашего пакетного менеджера успешно найдены файлы текстового редактора nano версии 9.0-1.)

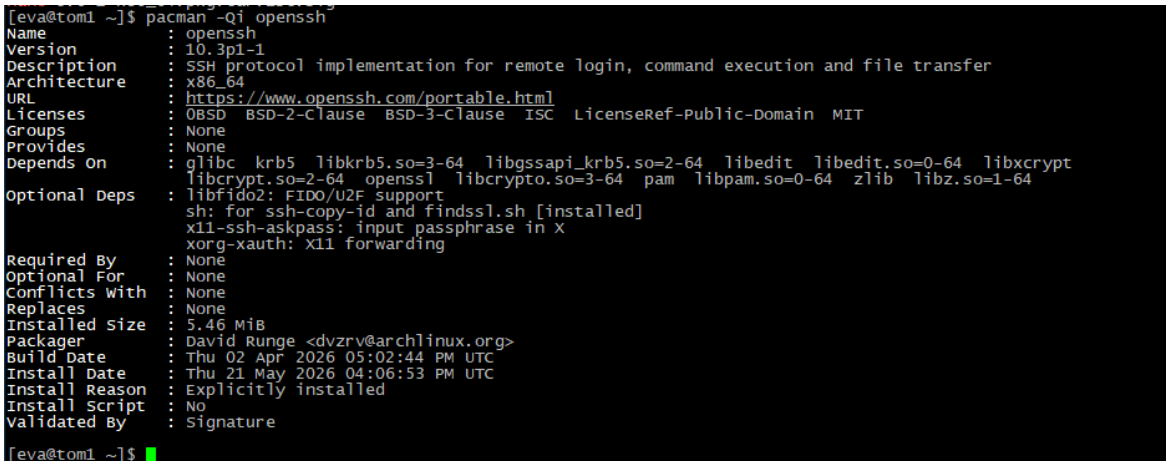
4.2. Пакет OpenSSH

Пакет OpenSSH в Arch Linux — это набор программ для безопасного удаленного доступа и управления компьютером по сети (через протокол SSH).

4.2.1. Проверка пакета openssh в системе

#bash

```
pacman -Qi openssh
```



(В выводе отобразилась подробная информация об уже установленном пакете openssh.)

4.2.2. Проверка пакета openssh в /var/cache/pacman/pkg/

Проверить наличие кэшированных файлов пакета nano можно с помощью стандартных команд терминала для поиска файлов. pacman хранит в этом каталоге скачанные архивные файлы с расширением .pkg.tar.zst (или .pkg.tar.xz).

#bash

```
ls /var/cache/pacman/pkg/ | grep openssh
```

(Этот вывод означает, что в кэше вашего пакетного менеджера успешно найдены файлы пакета openssh версии 10.3p1-1-x86_64.)

4.3. Пакет Samba

Пакет samba в Arch Linux — это реализация сетевого протокола SMB/CIFS. Он позволяет обмениваться файлами и принтерами между операционными системами Linux, Windows и macOS.

4.3.1. Проверка пакета samba в системе

#bash

```
pacman -Qi samba
```

```
[eva@tom1 ~]$ pacman -Qi samba
error: package 'samba' was not found
[eva@tom1 ~]$
```

(Этот вывод означает, что пакет samba в данный момент не установлен в вашей системе.)

4.3.2. Установка пакета samba

Скачивание и интеграция пакета в систему с автоматическим подтверждением (флаг –noconfirm) и пропуском уже установленных актуальных версий (–needed).

#bash

```
sudo pacman -S --noconfirm --needed samba
```

```
[eva@tom1 ~]$ sudo pacman -S --noconfirm --needed samba
[sudo] password for eva:
resolving dependencies...
looking for conflicting packages...

Packages (15) avahi-1:0.9rc4-1  cifs-utils-7.5-1  ldb-2:4.24.2-1  libbsd-0.12.2-2  libcups-2:2.4.19-1  libmd-1.2.0-1
              liburing-2.14-1  libwbclient-2:4.24.2-1  mpdecimal-4.0.1-3  python-3.14.5-1  smbclient-2:4.24.2-1
              talloc-2.4.4-1  tdb-1.4.15-1  tevent-1:0.17.1-2  samba-2:4.24.2-1

Total Download Size:    31.57 MiB
Total Installed Size:  173.03 MiB

:: Proceed with installation? [Y/n]
:: Retrieving packages...
 avahi-1:0.9rc4-1-x86_64                444.2 KiB  1136 KiB/s 00:00 [#####################################] 100%
 ldb-2:4.24.2-1-x86_64                  488.0 KiB  1117 KiB/s 00:00 [#####################################] 100%
 libcups-2:2.4.19-1-x86_64              273.4 KiB   899 KiB/s 00:00 [#####################################] 100%
 liburing-2.14-1-x86_64                 253.9 KiB   350 KiB/s 00:01 [#####################################] 100%
 libbsd-0.12.2-2-x86_64                 163.5 KiB   641 KiB/s 00:00 [#####################################] 100%
 mpdecimal-4.0.1-3-x86_64               103.2 KiB   593 KiB/s 00:00 [#####################################] 100%
 samba-2:4.24.2-1-x86_64                  8.7 MiB  5.41 MiB/s 00:02 [#####################################] 100%
 cifs-utils-7.5-1-x86_64                 98.7 KiB   914 KiB/s 00:00 [#####################################] 100%
 tdb-1.4.15-1-x86_64                     72.0 KiB   680 KiB/s 00:00 [#####################################] 100%
 libmd-1.2.0-1-x86_64                    59.8 KiB   672 KiB/s 00:00 [#####################################] 100%
 tevent-1:0.17.1-2-x86_64                58.2 KiB   710 KiB/s 00:00 [#####################################] 100%
 talloc-2.4.4-1-x86_64                   48.6 KiB   546 KiB/s 00:00 [#####################################] 100%
 libwbclient-2:4.24.2-1-x86_64           37.8 KiB   485 KiB/s 00:00 [#####################################] 100%
 python-3.14.5-1-x86_64                   2.0 MiB   882 KiB/s 00:13 [#####-------------------------------]  15%
 smbclient-2:4.24.2-1-x86_64              6.8 MiB  2.81 MiB/s 00:00 [##################################---]  93%
 Total (13/15)                           19.8 MiB  4.94 MiB/s 00:02 [#######################---------------]  62%
```

4.3.3. Проверка установки пакета samba

Верификация целостности установленных файлов пакета и их наличия в системе.

#bash

```
pacman -Qk samba
```

(Этот вывод означает, что пакет samba успешно установлен в системе, а все его файлы находятся на своих местах и не повреждены.)

Разбор строки вывода:

- **1195 total files** — пакет Samba содержит в себе ровно 1195 файлов.
- **0 missing files** — утерянных или отсутствующих файлов нет. Все 1195 компонентов успешно найдены в системе.

4.3.4. Проверка пакета samba в /var/cache/pacman/pkg/

Проверить наличие кэшированных файлов пакета nano можно с помощью стандартных команд терминала для поиска файлов. pacman хранит в этом каталоге скачанные архивные файлы с расширением .pkg.tar.zst (или .pkg.tar.xz).

#bash

```
ls /var/cache/pacman/pkg/ | grep samba
```

(Этот вывод означает, что в локальном кэше вашего пакетного менеджера успешно сохранен установочный архив пакета samba версии 4.24.2-1.)

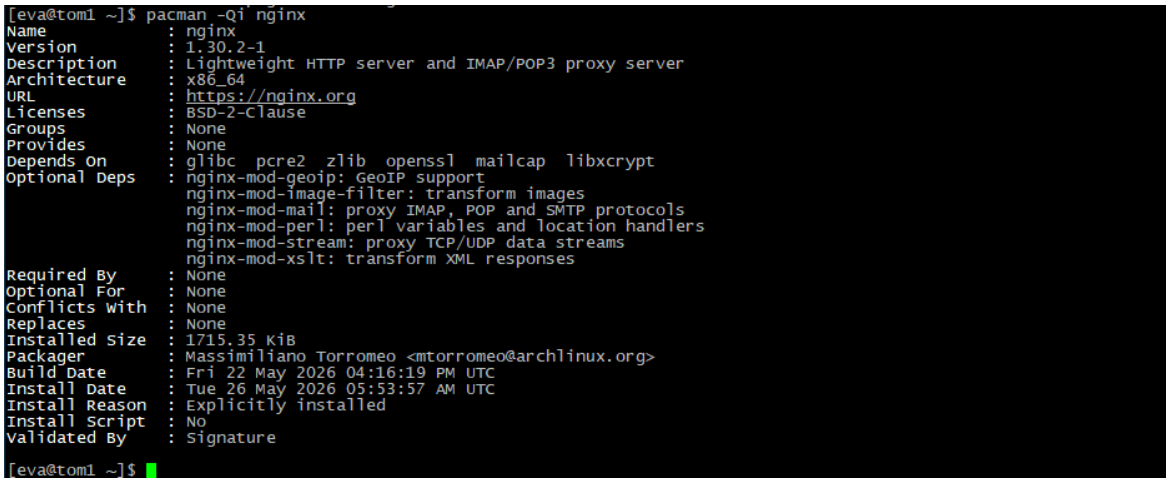
4.4. Пакет nginx

Пакет nginx в Arch Linux — это стандартный пакет, содержащий популярный высокопроизводительный веб-сервер и обратный прокси-сервер. Он используется для обработки HTTP-запросов, раздачи статических файлов, кэширования и перенаправления трафика.

4.4.1. Проверка пакета nginx в системе

#bash

```
pacman -Qi nginx
```



(Этот вывод содержит подробные метаданные об уже установленном в вашей системе веб-сервере nginx)

4.4.2.Проверка установки пакета nginx

Верификация целостности установленных файлов пакета и их наличия в системе.

#bash

```
pacman -Qk nginx
```

(Этот вывод означает, что пакет nginx успешно установлен в системе, а все его файлы находятся на своих местах и не повреждены.)

Разбор строки вывода:

- **47 total files** — пакет nginx содержит в себе ровно 47 файлов.
- **0 missing files** — утерянных или отсутствующих файлов нет. Все 47 компонентов успешно найдены в системе.

4.4.3. Проверка пакета nginx в /var/cache/pacman/pkg/

Проверить наличие кэшированных файлов пакета nginx можно с помощью стандартных команд терминала для поиска файлов. pacman хранит в этом каталоге скачанные архивные файлы с расширением .pkg.tar.zst (или .pkg.tar.xz).

#bash

```
ls /var/cache/pacman/pkg/ | grep nginx
```

(Этот вывод означает, что в кэше сохранены установочные файлы для двух разных версий веб-сервера nginx. Предыдущая версия (1.30.1). Текущая установленная версия (1.30.2))

4.5. Пакет php-fpm

В Arch Linux пакет php-fpm (FastCGI Process Manager) — это официальный компонент, который отвечает за обработку PHP-кода на сервере. По сути, это независимый менеджер фоновых процессов, который принимает запросы от веб-сервера (например, Nginx или Apache), выполняет PHP-скрипты и возвращает готовые страницы.

4.5.1. Проверка пакета php-fpm в системе

#bash

```
pacman -Qi php-fpm
```

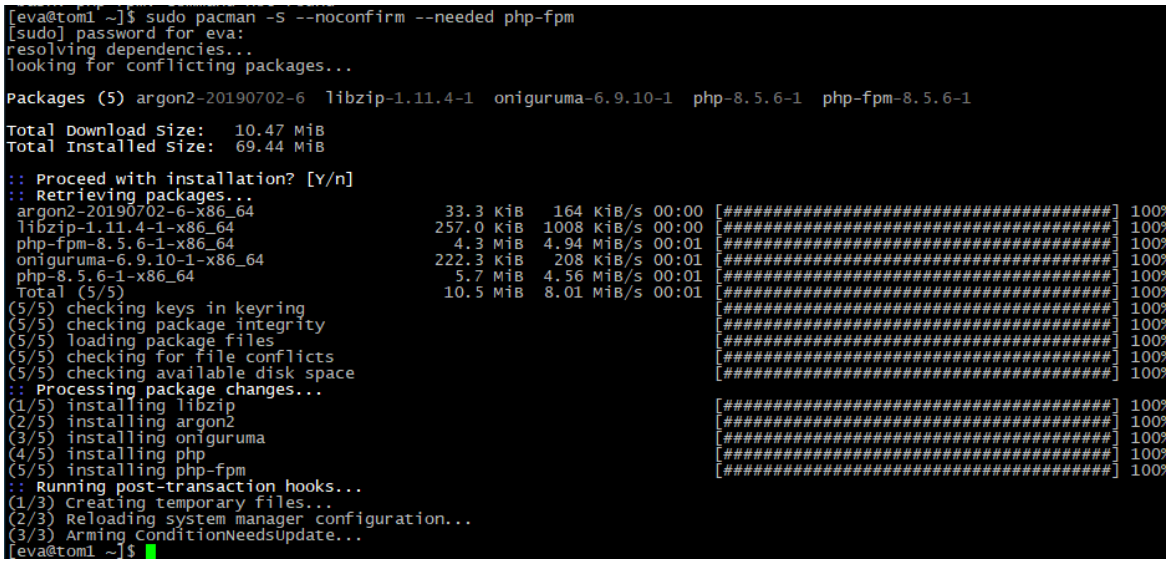
(Этот вывод означает, что пакет php-fpm в данный момент не установлен в вашей системе.)

4.5.2. Установка пакета php-fpm

Скачивание и интеграция пакета в систему с автоматическим подтверждением (флаг –noconfirm) и пропуском уже установленных актуальных версий (–needed).

#bash

```
sudo pacman -S --noconfirm --needed php-fpm
```



4.5.3. Проверка установки пакета php-fpm

Верификация целостности установленных файлов пакета и их наличия в системе.

#bash

```
pacman -Qk php-fpm
```

(Этот вывод означает, что пакет php-fpm успешно установлен в системе, а все его файлы находятся на своих местах и не повреждены.)

Разбор строки вывода:

- **21 total files** — пакет php-fpm содержит в себе ровно 21 файлов.
- **0 missing files** — утерянных или отсутствующих файлов нет. Все 21 компонентов успешно найдены в системе.

4.5.4. Проверка пакета php-fpm в /var/cache/pacman/pkg/

Проверить наличие кэшированных файлов пакета php-fpm можно с помощью стандартных команд терминала для поиска файлов. pacman хранит в этом каталоге скачанные архивные файлы с расширением .pkg.tar.zst (или .pkg.tar.xz).

#bash

```
ls /var/cache/pacman/pkg/ | grep php-fpm
```

(Этот вывод означает, что в кэше пакетов сохранен готовый к установке архив php-fpm версии 8.5.6-1.)

4.6. Пакет squashfs-tools

Пакет squashfs-tools в Arch Linux — это набор утилит командной строки для создания, распаковки и модификации сжатых файловых систем SquashFS.

4.6.1. Проверка пакета squashfs-tools в системе

#bash

```
pacman -Qi squashfs-tools
```

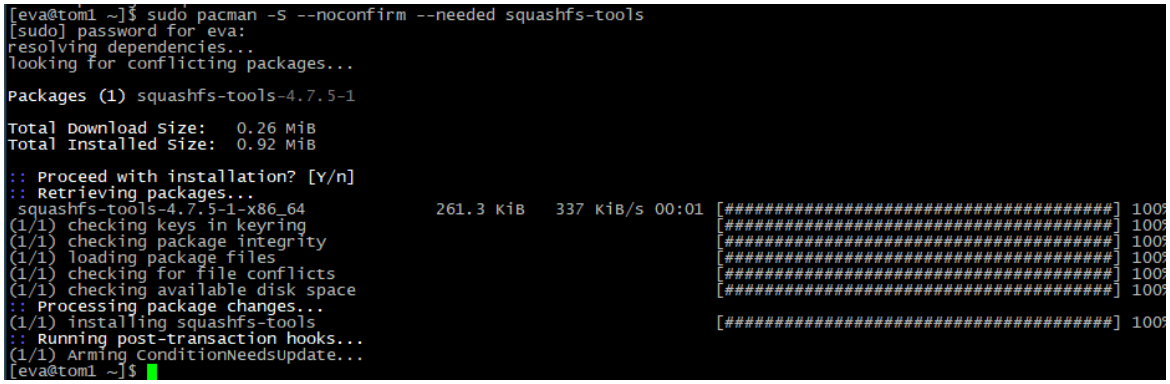
(Этот вывод означает, что пакет squashfs-tools в данный момент не установлен в вашей системе.)

4.6.2. Установка пакета squashfs-tools

Скачивание и интеграция пакета в систему с автоматическим подтверждением (флаг –noconfirm) и пропуском уже установленных актуальных версий (–needed).

#bash

```
sudo pacman -S --noconfirm --needed squashfs-tools
```



4.6.3. Проверка установки пакета squashfs-tools

Верификация целостности установленных файлов пакета и их наличия в системе.

#bash

```
pacman -Qk squashfs-tools
```

(Этот вывод означает, что пакет squashfs-tools успешно установлен в системе, а все его файлы находятся на своих местах и не повреждены.)

Разбор строки вывода:

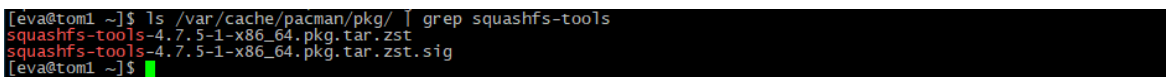
- **31 total files** — пакет squashfs-tools содержит в себе ровно 31 файлов.
- **0 missing files** — утерянных или отсутствующих файлов нет. Все 31 компонентов успешно найдены в системе.

4.6.4. Проверка пакета squashfs-tools в /var/cache/pacman/pkg/

Проверить наличие кэшированных файлов пакета squashfs-tools можно с помощью стандартных команд терминала для поиска файлов. pacman хранит в этом каталоге скачанные архивные файлы с расширением .pkg.tar.zst (или .pkg.tar.xz).

#bash

```
ls /var/cache/pacman/pkg/ | grep squashfs-tools
```



(Этот вывод означает, что в локальном кэше вашего пакетного менеджера сохранен установочный архив пакета squashfs-tools версии 4.7.5-1.)

4.7. Пакет zram-generator

zram-generator в Arch Linux — это компонент, который автоматически создает сжатые блочные устройства в оперативной памяти (RAM) и настраивает их в качестве пространства подкачки (swap) с помощью системного менеджера systemd.

4.7.1. Проверка пакета zram-generator в системе

#bash

```
pacman -Qi zram-generator
```

(Этот вывод содержит подробные метаданные об уже установленном в вашей системе пакете zram-generator версии 1.2.1-1.)

4.7.2. Проверка установки пакета zram-generator

Верификация целостности установленных файлов пакета и их наличия в системе.

#bash

```
pacman -Qk zram-generator
```

(Этот вывод означает, что пакет zram-generator полностью целостен и все его компоненты находятся на своих местах в операционной системе.)

Разбор строки вывода:

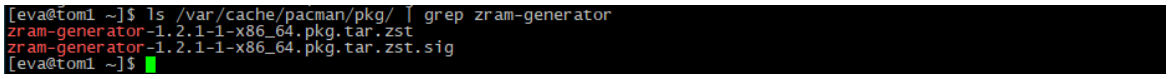
- **19 total files** — пакет содержит в себе ровно 19 файлов (исполняемые файлы, файлы конфигурации systemd, документация).
- **0 missing files** — утерянных, поврежденных или случайно удаленных файлов не обнаружено. Все 19 компонентов успешно найдены в системе.

4.7.3. Проверка пакета zram-generator в /var/cache/pacman/pkg/

Проверить наличие кэшированных файлов пакета zram-generator можно с помощью стандартных команд терминала для поиска файлов. pacman хранит в этом каталоге скачанные архивные файлы с расширением .pkg.tar.zst (или .pkg.tar.xz).

#bash

```
ls /var/cache/pacman/pkg/ | grep zram-generator
```



(Этот вывод означает, что в локальном кэше пакетов сохранен установочный архив версии пакета zram-generator, которая сейчас установлена у вас в системе.)

4.8. Утилита xorriso

Утилита xorriso в Arch Linux — это мощная консольная программа для создания, изменения и извлечения ISO-образов (файловых систем ISO 9660 с расширениями Rock Ridge), а также для записи данных и образов на оптические диски (CD, DVD, Blu-ray).

4.8.1. Проверка утилиты xorriso в системе

#bash

```
pacman -Qi xorriso
```

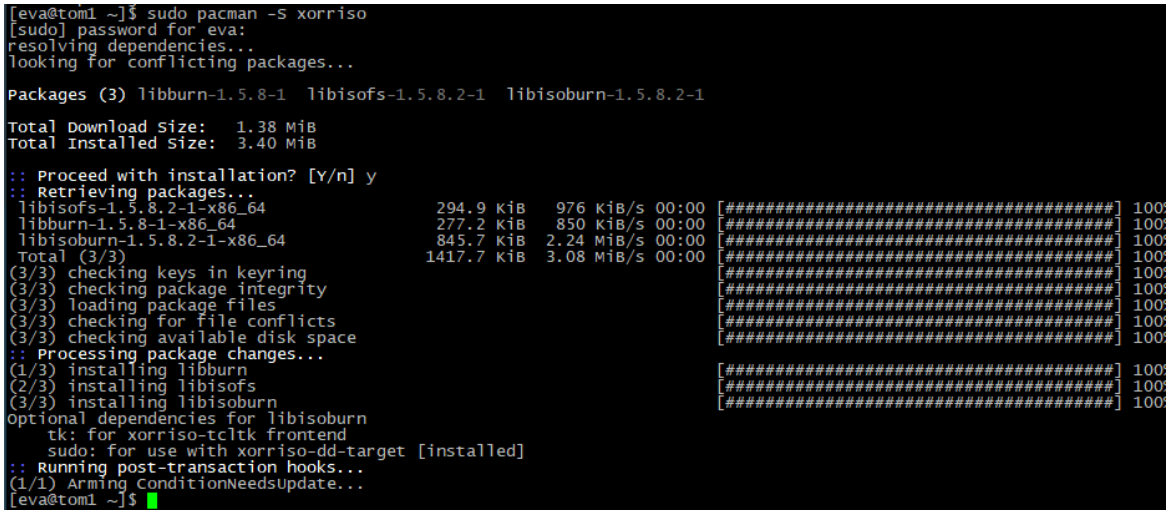
(Этот вывод означает, что утилита xorriso в данный момент не установлена в вашей системе.)

4.8.2. Установка утилиты xorriso

#bash

```
sudo pacman -S xorriso
```

(Система запросит подтверждение установки, нажмите Y и Enter).



(Этот вывод означает, что вы успешно установили зависимости утилиты xorriso, являющтхся её основой)
Менеджер пакетов pacman перед установкой xorriso установил необходимые для его работы библиотеки (libburn, libisofs, libisoburn).

4.8.3. Проверка установки утилиты xorriso

Верификация целостности установленных файлов пакета и их наличия в системе.

#bash

```
pacman -Qk xorriso
```

(Этот вывод наглядно подтверждает, что утилита xorriso физически находится внутри пакета libisoburn, и все её файлы полностью целы.)

Разбор строки вывода:

- **libisoburn: 32 total files** — pacman автоматически перенаправил запрос на реальный пакет libisoburn, так как именно он отвечает за эту программу в Arch Linux.).
- **0 missing files** — все 32 файла (включая саму команду xorriso) успешно найдены на вашем жестком диске. Ни один компонент не утерян и не.

4.8.4. Проверка пакета xorriso в /var/cache/pacman/pkg/

Проверить наличие кэшированных файлов пакета xorriso можно с помощью стандартных команд терминала для поиска файлов. pacman хранит в этом каталоге скачанные архивные файлы с расширением .pkg.tar.zst (или .pkg.tar.xz).
Т.к. реальное имя пакета в системе — libisoburn, чтобы правильно проверить наличие установочных файлов в кэше, выполните поиск по ключевому слову isoburn:

#bash

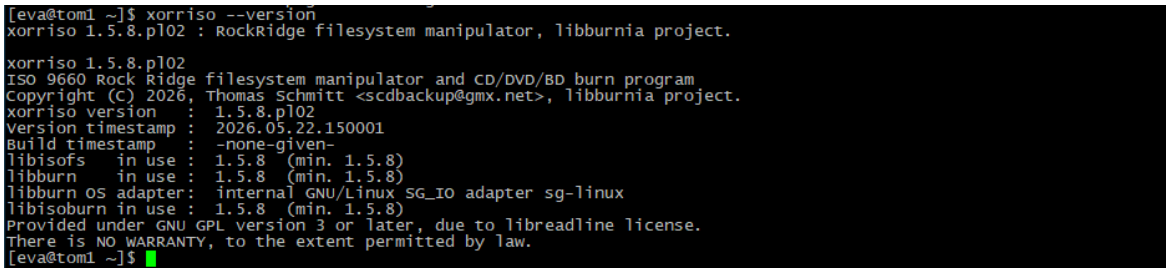
```
ls /var/cache/pacman/pkg/ | grep isoburn
```

(Этот вывод означает, что в локальном кэше вашего пакетного менеджера успешно сохранен установочный архив пакета libisoburn (который и содержит в себе утилиту xorriso).)

4.8.5. Проверяем версию установленной утилиты xorriso

#bash

```
xorriso --version
```



(Этот вывод означает, что утилита xorriso успешно запущена, полностью исправна и готова к работе.)

5. Установка дополнительных пакетов

В предыдущей главе, мы проверили и установили все основные пакеты и утилиты, необходимые нам для сборки кастомного iso образа сервера Arch linux, который будут собран и записан на физический USB - флеш накопитель, для установки системы на локальном сервере (Offline mode) в изолированной сети (air-gapped network).

Так как цель данного руководства - сборка полностью автомомного сервера с web-приложением для настройки сервера под любые задачи, мы загрузим в кэш pacman (/var/cache/pacman/pkg/) дополнительные пакеты, перечень которых вы можите изменить по своему усмотрению. (см. список всех пакетов устанавливаемых при выполнении этого пункта).

5.1. Сохранение в кэш дополнительного ПО

Используем для этой цели скрипт, обернутый в subo cat « 'EOF' для загрузки дополнительный пакетов с зависимостями в arch linux в /var/cache/pacman/pkg/ с автоподтверждением и проверкой целосности каждого пакета, без установки в систему, с выводом в конце списка загруженных пакетов
Чтобы запустить весь этот процесс напрямую в консоли, скопируйте весь блок кода ниже, вставьте его в свой терминал и нажмите Enter. Команда сразу запросит пароль администратора и начнет выполнение:

#bash

```
sudo bash << 'EOF'
# Полный список ваших пакетов
PACKAGES=(
    btrfs-progs e2fsprogs mdadm lvm2 snapper inxi man-pages texinfo
    networkmanager dnsmasq
    pam libpam-google-authenticator qrencode ufw fail2ban certbot
clamav
    apache php-apache mariadb phpmyadmin php
    nfs-utils bftpd rsync
    dokuwiki wordpress gitea
    minidlna mpd immich-go radicale opensmtpd
    code python feh libreoffice-fresh
    docker qemu-system-x86 kcron fzy
)

CACHE_DIR="/var/cache/pacman/pkg"

echo "=== 1. Синхронизация баз данных пакетов ==="
pacman -Sy --noconfirm

echo -e "\n=== 2. Загрузка пакетов и их зависимостей (без установки) ==="
pacman -Sw --noconfirm --needed "${PACKAGES[@]}"
# -S: установка/загрузка
# -w: только скачивание (без установки)
```

```
# --noconfirm: автоподтверждение всех запросов
# --needed: не скачивать заново то, что уже есть актуального в кэше/системе

echo -e "\n=== 3. Проверка целостности архивов и сбор версий ==="
echo "-------------------------------------------------------------------
-----"
printf "%-30s %-20s %s\n" "ПАКЕТ" "ВЕРСИЯ" "СТАТУС КЭША"
echo "-------------------------------------------------------------------
-----"

ERRORS=0

for pkg in "${PACKAGES[@]}"; do
  # Получаем версию пакета из базы данных pacman
    VERSION=$(pacman -Si "$pkg" | awk -F': ' '/^Version/ {print $2}' |
xargs)

    if [ -z "$VERSION" ]; then
        printf "%-30s %-20s %s\n" "$pkg" "---" "[НЕ НАЙДЕН В РЕПО]"
        ((ERRORS++))
        continue
    fi

  # Очищаем версию от эпохи (двоеточия) для сопоставления с именем файла
    CLEAN_VER=$(echo "$VERSION" | sed 's/^[0-9]://')

  # Поиск архива в кэше загрузок
    PKG_FILE=$(ls "$CACHE_DIR" 2>/dev/null | grep -E "^${pkg}-
${CLEAN_VER}-(x86_64|any)\.pkg\.tar\.zst$")

    if [ -n "$PKG_FILE" ]; then
    # Проверка целостности структуры сжатого архива утилитой zstd
        if zstd -t "$CACHE_DIR/$PKG_FILE" &>/dev/null; then
            STATUS="[OK] Архив цел"
        else
            STATUS="[ОШИБКА] Поврежден"
            ((ERRORS++))
        fi
    else
        STATUS="[ОТСУТСТВУЕТ]"
        ((ERRORS++))
    fi

    printf "%-30s %-20s %s\n" "$pkg" "$VERSION" "$STATUS"
done

echo "-------------------------------------------------------------------
-----"
echo "Процесс завершен. Проблемных пакетов: $ERRORS"
EOF
```

```
> echo "------------------------------------------------------------------------"
> echo "Процесс завершен. Проблемных пакетов: $ERRORS"
> EOF
[sudo] password for eva:
=== 1. Синхронизация баз данных пакетов ===
:: Synchronizing package databases...
 core                                  126.7 KiB   587 KiB/s 00:00 [#####################################] 100%
 extra                                   8.2 MiB  8.35 MiB/s 00:01 [#####################################] 100%

=== 2. Загрузка пакетов и их зависимостей (без установки) ===
warning: btrfs-progs-7.0-1 is up to date -- skipping
warning: e2fsprogs-1.47.4-1 is up to date -- skipping
warning: networkmanager-1.56.1-1 is up to date -- skipping
warning: pam-1.7.2-2 is up to date -- skipping
warning: php-8.5.6-1 is up to date -- skipping
warning: python-3.14.5-1 is up to date -- skipping
resolving dependencies...
:: There are 2 providers available for jack:
:: Repository extra
   1) jack2  2) pipewire-jack

Enter a number (default=1):
:: There are 2 providers available for cron:
```

В окне терминала пойдет загрузка перечня загружаемых пакетом, с выводом пакета, версии и проверкой целосности пакета

```
=== 3. Проверка целостности архивов и сбор версий ===
------------------------------------------------------------------------
ПАКЕТ                       ВЕРСИЯ          СТАТУС КЭША
------------------------------------------------------------------------
btrfs-progs                 7.0-1           [OK] Архив цел
e2fsprogs                   1.47.4-1        [OK] Архив цел
mdadm                       4.6-2           [OK] Архив цел
lvm2                        2.03.41-1       [OK] Архив цел
snapper                     0.13.1-2        [OK] Архив цел
inxi                        3.3.40.1-1      [OK] Архив цел
man-pages                   6.18-1          [OK] Архив цел
texinfo                     7.3-1           [OK] Архив цел
networkmanager              1.56.1-1        [OK] Архив цел
dnsmasq                     2.92.rel2-2     [OK] Архив цел
pam                         1.7.2-2         [OK] Архив цел
libpam-google-authenticator 1.11-1          [OK] Архив цел
qrencode                    4.1.1-4         [OK] Архив цел
ufw                         0.36.2-7        [OK] Архив цел
fail2ban                    1.1.0-8         [OK] Архив цел
certbot                     5.6.0-1         [OK] Архив цел
clamav                      1.5.2-2         [OK] Архив цел
apache                      2.4.67-1        [OK] Архив цел
php-apache                  8.5.6-1         [OK] Архив цел
mariadb                     12.2.2-2        [OK] Архив цел
phpmyadmin                  5.2.3-1         [OK] Архив цел
php                         8.5.6-1         [OK] Архив цел
nfs-utils                   2.9.1-1         [OK] Архив цел
bftpd                       6.6-1           [OK] Архив цел
rsync                       3.4.3-1         [OK] Архив цел
dokuwiki                    20250514_b-1    [OK] Архив цел
wordpress                   7.0-1           [OK] Архив цел
gitea                       1.26.2-1        [OK] Архив цел
minidlna                    1.3.3-6         [OK] Архив цел
mpd                         0.24.12-1       [OK] Архив цел
immich-go                   1:0.31.0-1      [ОТСУТСТВУЕТ]
radicale                    3.7.3-1         [OK] Архив цел
opensmtpd                   7.8.0p1-1       [OK] Архив цел
code                        1.121.0-1       [OK] Архив цел
python                      3.14.5-1        [OK] Архив цел
feh                         3.12.2-1        [OK] Архив цел
libreoffice-fresh           26.2.3-3        [OK] Архив цел
docker                      1:29.5.1-1      [ОТСУТСТВУЕТ]
qemu-system-x86             11.0.0-1        [OK] Архив цел
kcron                       26.04.1-1       [OK] Архив цел
fzy                         1.1-1           [OK] Архив цел
------------------------------------------------------------------------
Процесс завершен. Проблемных пакетов: 2
[eva@tom1 ~]$
```

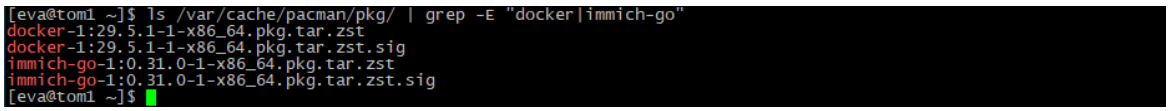
(Этот вывод означает, что ваш скрипт успешно отработал и 95% запрошенных пакетов были успешно загружены и проверены в кэше.)
Однако два пакета (docker и immich-go) не смогли скачаться, из-за чего в самом низу лога вывелось сообщение: «Процесс завершен. Проблемных пакетов: 2».
При этом прокрутив ползунок кансоли, мы убеждаемся, что пакет docker загружен в кэш.

```
 mariadb-12.2.2-2-x86_64                29.1 MiB  1775 KiB/s 00:17 [#####################################] 100%
 docker-1:29.5.1-1-x86_64               27.5 MiB  1585 KiB/s 00:18 [#####################################] 100%
 wordpress-7.0-1-any                    22.9 MiB  1539 KiB/s 00:15 [#####################################] 100%
```

5.2. Проверка docker и immich-go в кэш

#bash

```
ls /var/cache/pacman/pkg/ | grep -E "docker|immich-go"
```



(Этот вывод означает, что пакеты docker и immich-go успешно скачаны и находятся в локальном кэше вашего менеджера пакетов.)

6. Обновление баз, кэша и системы

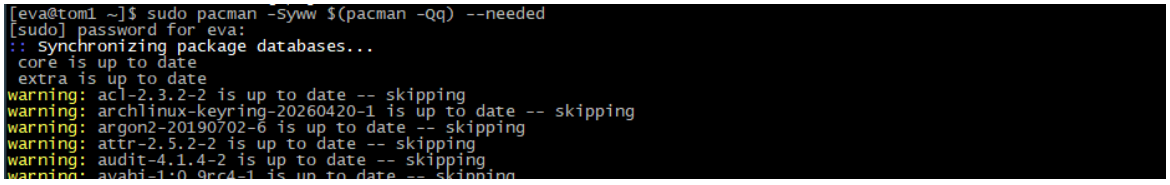
6.1. Обновление баз и загрузка всех зависимостей в кэш

Эта команда берет список всех файлов из вашей папки /var/cache/pacman/pkg/, сверяет их с сервером и скачивает их новые версии в кэш (в систему ничего не ставится), а так же автоматически скачивает абсолютно все необходимые зависимости:

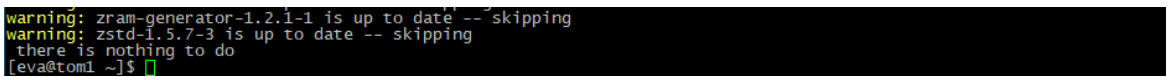
#bash

```
sudo pacman -Syww $(pacman -Qq) --neede
```

(Использование флагов -S (синхронизация), -y (обновление баз данных репозиториев) и -w (только скачивание))



(Вывод на скрине ниже означает, что команда отработала идеально и ваш кэш уже находится в максимально актуальном состоянии.)



6.1.1. Получение общего размера кэша на диске:

#bash

```
du -sh /var/cache/pacman/pkg/
```

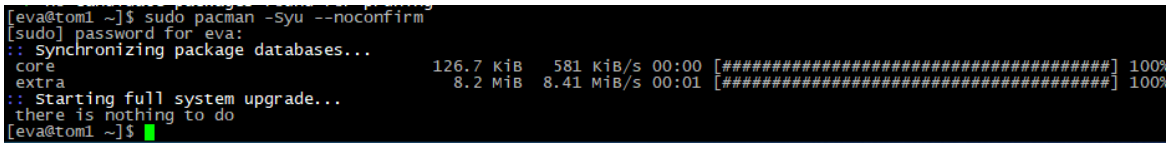
6.2. Обновление установленных пакетов и зависимостей в системе

Чтобы обновить абсолютно все установленные пакеты в системе вместе с их зависимостями и при этом автоматически подтверждать все запросы (чтобы команда не останавливалась и не задавала вопросов), выполните:

#bash

```
sudo pacman -Syu --noconfirm
```

*(**-Syu** — синхронизирует базы данных репозиториев (**-y**) и запускает полный апгрейд всех устаревших пакетов в системе (**-u**) вместе с их зависимостями. **–noconfirm** — главный флаг автоматизации. Он отключает все всплывающие вопросы вида «Хотите продолжить установку? [Y/n]» и автоматически выбирает вариант Да (Yes).)*



(Этот вывод означает, что ваша система уже полностью обновлена, и устанавливать ничего не нужно.)

7. Статический IP адрес хоста

Вместо автоматического DHCP мы жестко (статически) пропишем адрес к примеру 192.168.1.72 в файле конфигурации. Это гарантирует, что хост tom_1 больше никогда не сменит свой IP, а сетевой диск в Windows не отвалится.
Чтобы настроить статический IP-адрес в Arch Linux через systemd-networkd, нам нужно создать или отредактировать имеющийся файл конфигурационный файл с расширением .network

7.1. Проверка директории /etc/systemd/network/

#bash

```
ls -la /etc/systemd/network/
```

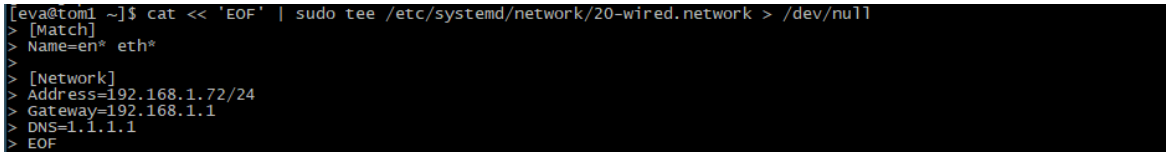
(Если в выводе будут только строки . и .., значит, папка полностью пуста.)

7.2. Если конфигурационного файла .network нет

Создайте файл с расширением .network. Имя файла может быть любым, но оно должно начинаться с цифр (чтобы задать приоритет).

#bash

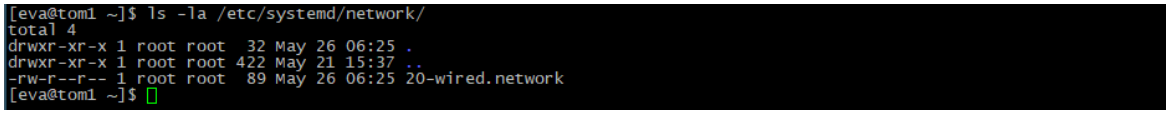
```
cat << 'EOF' | sudo tee /etc/systemd/network/20-wired.network >
/dev/null
[Match]
Name=en* eth*

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



7.3. Если конфигурационный файл .network есть

#bash

```
ls -la /etc/systemd/network/
```



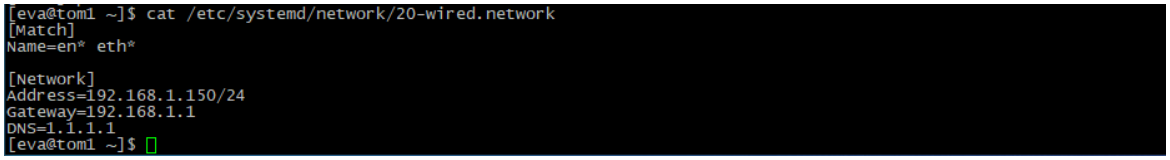
(Имя файла определено: 20-wired.network.)

Перед тем как вносить изменения, мы обязаны посмотреть его полную структуру, чтобы увидеть секцию [Match] (которая привязывает этот конфиг к конкретной сетевой карте eth0 или enx...).

7.3.1. Структура файла 20-wired.network

#bash

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



(Секция [Match] перехватывает все интерфейсы, начинающиеся на en и eth*.)*
Теперь мы готовы переписать этот файл, заменив красивый адрес 192.168.1.150 на ваш более статический 192.168.1.72. Остальные параметры (маску /24, шлюз и DNS) оставляем без изменений. *(а адрес 192.168.1.150 мы оставим для нашей флешки с iso образом для обнаружения в сети)*

7.3.2. Перезапись файла 20-wired.network на IP 192.168.1.72

Мы используем монолитную команду cat « 'EOF', которая полностью затрёт старый конфиг и запишет новый чистый текст. Это исключает ошибки ручного ввода в редакторах.
Выполните в терминале команду:

#bash

```
cat << 'EOF' | sudo tee /etc/systemd/network/20-wired.network >
/dev/null
[Match]
Name=en* eth*

[Network]
Address=192.168.1.72/24
Gateway=192.168.1.1
DNS=1.1.1.1
EOF
```

```
[eva@tom1 ~]$ cat << 'EOF' | sudo tee /etc/systemd/network/20-wired.network > /dev/null
> [Match]
> Name=en* eth*
>
> [Network]
> Address=192.168.1.72/24
> Gateway=192.168.1.1
> DNS=1.1.1.1
> EOF
[sudo] password for eva:
[eva@tom1 ~]$
```

Перед тем как перезапустить сеть, мы обязаны сделать шаг контроля и убедиться, что файл перезаписался именно так, как нам нужно.

7.3.3. Проверка измененного файла 20-wired.network

#bash

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

```
[eva@tom1 ~]$ cat /etc/systemd/network/20-wired.network
[Match]
Name=en* eth*

[Network]
Address=192.168.1.72/24
Gateway=192.168.1.1
DNS=1.1.1.1
[eva@tom1 ~]$
```

(Внутри прописан наш целевой статический адрес 192.168.1.72.)

7.3.4. Применение настроек сети

Чтобы система сбросила старый адрес 192.168.1.150 и применила новый, нам необходимо полностью перезапустить сетевую службу systemd-networkd.

#bash

```
sudo systemctl restart systemd-networkd
```

Важно: Как только вы выполните эту команду, текущая SSH-сессия PuTTY сразу же прервётся (окно зависнет), так как IP-адрес машины мгновенно изменится на 192.168.1.72.

```
[eva@tom1 ~]$ sudo systemctl restart systemd-networkd
[sudo] password for eva:
```

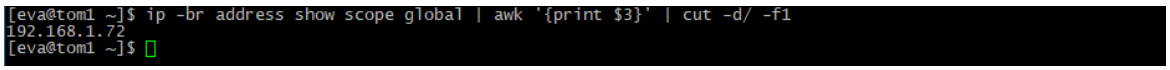
7.3.5. Проверка нового адреса

Откройте новое окно PuTTY и подключитесь к tom_1 по его новому постоянному адресу: 192.168.1.72

#bash

```
ip -br address show scope global | awk '{print $3}' | cut -d/ -f1
```

(Система отобразила наш новый постоянный IP-адрес 192.168.1.72)



8. Развертывание и настройка копоненонтов по WebUI

На нашем будущем сервере будет постоянно запущен веб-сервер Nginx. Он предоставит пользователю возможность управлять системой через веб-приложение в окне браузера по временному порту 7000. Если пользователь захочет использовать собственный веб-сервер, в приложении ему будет доступен веб-сервер Apache2 с поддержкой PHP. Службы OpenSSH и Samba, автозапуск которых настраивается для создания веб-приложения, также будут по умолчанию отключены с возможностью их постоянного включения через панель управления.

8.1. Настройка Nginx на обработку PHP

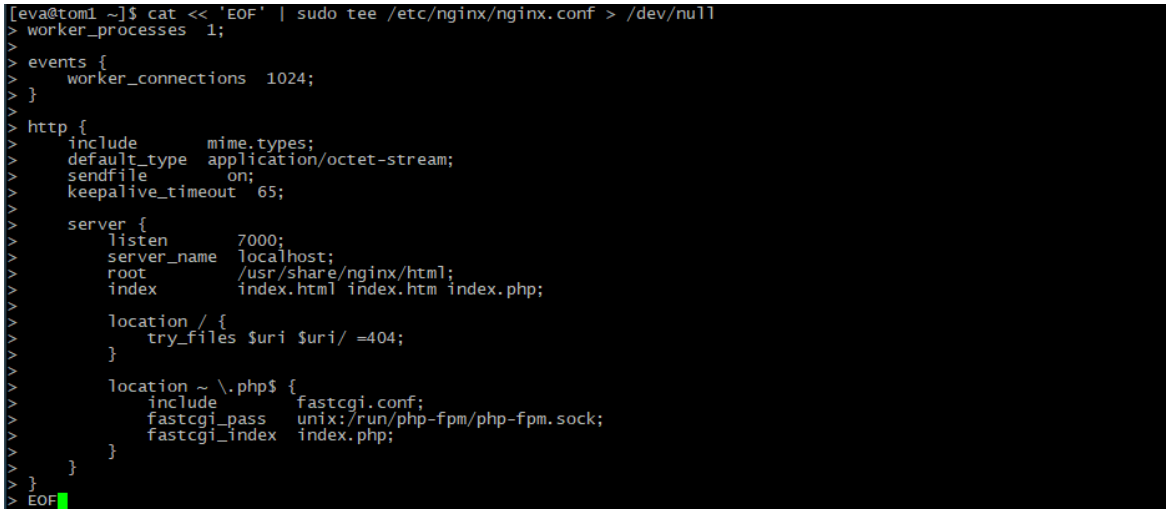
8.1.1. Файл nginx.conf

Мы перепишем конфигурационный файл nginx.conf, добавив правильный блок location ~ \.php$, работающий через UNIX-сокет.

#bash

```
cat << 'EOF' | sudo tee /etc/nginx/nginx.conf > /dev/null
worker_processes  1;

events {
    worker_connections  1024;
}

http {
    include       mime.types;
    default_type  application/octet-stream;
    sendfile        on;
    keepalive_timeout  65;

    server {
        listen       7000;
        server_name  localhost;
        root         /usr/share/nginx/html;
        index        index.html index.htm index.php;
```

```
        location / {
            try_files $uri $uri/ =404;
        }

        location ~ \.php$ {
            include        fastcgi.conf;
            fastcgi_pass   unix:/run/php-fpm/php-fpm.sock;
            fastcgi_index  index.php;
        }
    }
}
EOF
```

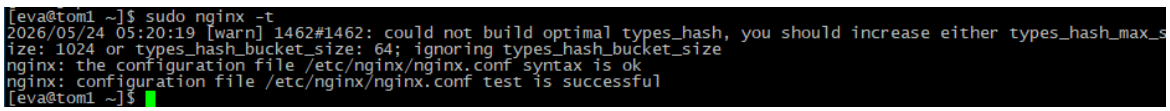


8.1.2. Тестирование синтаксиса nginx.conf

Нам нужно протестировать синтаксис Nginx, чтобы убедиться, что все скобки закрыты и сокет прописан без ошибок.

#bash

```
sudo nginx -t
```



(Тест пройден успешно (syntax is ok, test is successful). Предупреждение о types_hash — это стандартный информационный ворнинг, на работу он не влияет.)

8.1.3. Проверка содержимого файла nginx.conf

#bash

```
cat -n /etc/nginx/nginx.conf
```

8.1.4. Перезапуск веб-сервера

#bash

```
sudo systemctl restart nginx
```

8.1.5. Проверка статуса службы

#bash

```
sudo systemctl status nginx
```

```
[eva@tom1 ~]$ sudo systemctl status nginx
● nginx.service - nginx web server
     Loaded: loaded (/usr/lib/systemd/system/nginx.service; enabled; preset: disabled)
     Active: active (running) since Sun 2026-05-24 05:31:24 UTC; 2min 13s ago
 Invocation: c70deed23faa4bcca7e4807d5183a936
    Process: 1520 ExecStart=/usr/bin/nginx (code=exited, status=0/SUCCESS)
   Main PID: 1521 (nginx)
      Tasks: 2 (limit: 11644)
     Memory: 4.8M (peak: 5.3M)
        CPU: 16ms
     CGroup: /system.slice/nginx.service
             ├─1521 "nginx: master process /usr/bin/nginx"
             └─1522 "nginx: worker process"

May 24 05:31:24 tom1 systemd[1]: Stopping nginx web server...
May 24 05:31:24 tom1 systemd[1]: nginx.service: Deactivated successfully.
May 24 05:31:24 tom1 systemd[1]: Stopped nginx web server.
May 24 05:31:24 tom1 systemd[1]: Starting nginx web server...
May 24 05:31:24 tom1 nginx[1520]: 2026/05/24 05:31:24 [warn] 1520#1520: could not build optimal types_hash, you shou
d increase either types_hash_max_size: 1024 or types_hash_bucket_size: 64; ignoring types_hash_bucket_size
May 24 05:31:24 tom1 systemd[1]: Started nginx web server.
[eva@tom1 ~]$
```

8.1.6. Проверим директорию

Чтобы не запутаться в структуре бэкенда и фронтенда, давайте сначала проверим, что сейчас вообще находится внутри корневой папки Nginx на tom_1.

#bash

```
ls -la /usr/share/nginx/html/
```

```
[eva@tom1 ~]$ ls -la /usr/share/nginx/html/
total 8
drwxrwxrwx 1 root root  36 May 23 09:21 .
drwxr-xr-x 1 root root   8 May 21 16:06 ..
-rwxrwxrwx 1 root root 497 May 22 16:16 50x.html
-rwxrwxrwx 1 root root 896 May 22 16:16 index.html
```

Проверим запуск и работу веб-сервера на порту 7000 в браузере на странице http://192.168.1.72:7000/

8.2. Проверяем автозапуск службы php-fpm

Убедимся, что служба PHP-FPM активирована в автозапуске, чтобы при старте флешки в ОЗУ

она запустилась сама вместе с Nginx.

#bash

```
systemctl is-enabled php-fpm
```

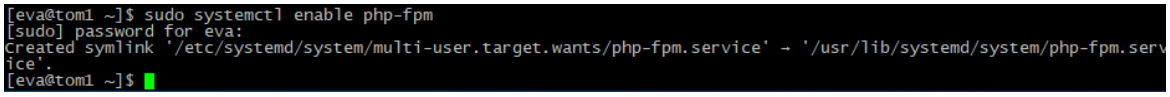
(Примечание: статус disabled - выключена)

8.2.1. Включаем службу php-fpm в автозапуск

Включим службу PHP-FPM в автозапуске, чтобы при старте флешки в ОЗУ она запустилась сама вместе с Nginx.

#bash

```
sudo systemctl enable php-fpm
```



(Примечание: созданы системные симлинки)

8.2.1.1. Проверяем статус автозапуска службы php-fpm

Убедимся, что служба PHP-FPM активирована в автозапуске, чтобы при старте флешки в ОЗУ она запустилась сама вместе с Nginx. Убедимся, что система теперь рапортует правильный статус.

#bash

```
systemctl is-enabled php-fpm
```

(Примечание: статус enabled - включена)

8.2.2. Снятие системной изоляции с PHP

В Arch Linux служба `php-fpm` по умолчанию заперта (ProtectSystem=full). Из-за этого PHP видит системные файлы пользователей как Read-Only. Снимем это ограничение.

Откройте переопределение настроек службы:

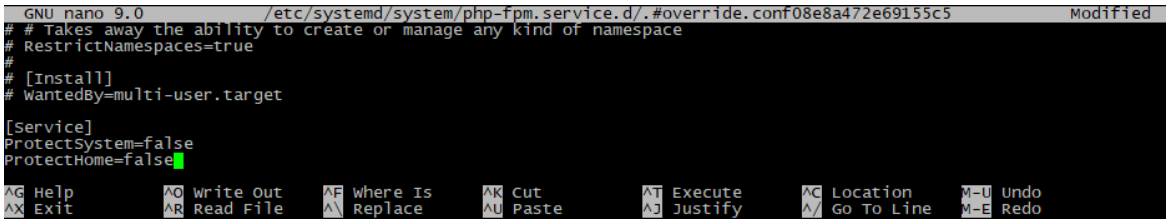
#bash

```
sudo systemctl edit php-fpm
```

Вставте свои строки строго между первой и второй строками комментариев (обычно там есть явная подсказка ### Lines below this comment will be discarded или аналогичная).:

ini

```
[Service]
ProtectSystem=false
ProtectHome=false
```



(Ctrl+O для сохранения, затем Enter и Ctrl+X для выхода)

(Вывод означает правильно отредактированную конфигурацию, и systemd успешно создал конфигурационный файл (drop-in файл) с вашими настройками по пути /etc/systemd/system/php-fpm.service.d/override.conf)

Перезапустите службу:

```
sudo systemctl restart php-fpm
```

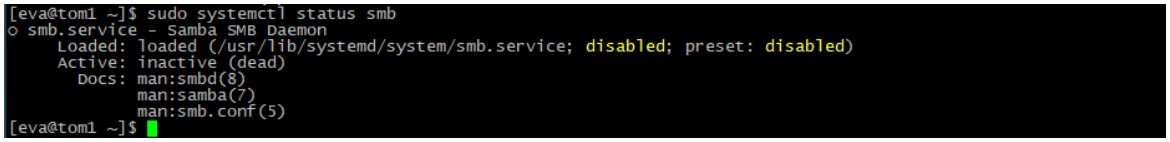
8.3. Настройка Samba-сервера

8.3.1. Проверяем состояние Samba-сервера на tom_1

Нам нужно узнать, запущен ли демон Samba (smb), чтобы понять, сможем ли мы сразу подключить сетевой диск в Windows.
Выполните в терминале команду:

#bash

```
sudo systemctl status smb
```



(Этот вывод означает, что служба установлена в системе, но её автозапуск отключен и в данный момент служба не запущена (остановлена))

8.3.2. Создание конфигурационного файла smb.conf

Создадим конфигурационный файл в редакторе nano.
Выполните в терминале команду:

#bash

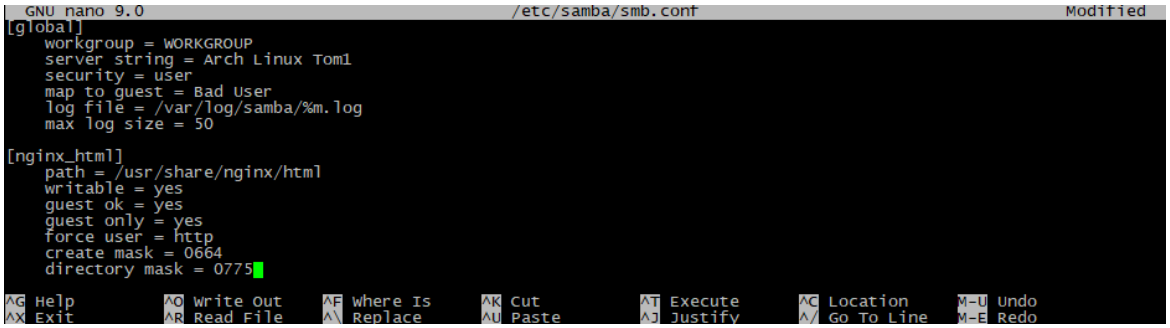
```
sudo nano /etc/samba/smb.conf
```

Файл пустой и готов к заполнению. Вставьте в него следующий минимальный рабочий конфиг, чтобы открыть доступ к директории Nginx (/usr/share/nginx/html) с автоматическим наследованием безопасных прав доступа (775 для папок и 664 для файлов):

ini

```
[global]
    workgroup = WORKGROUP
    server string = Arch Linux Tom1
    security = user
    map to guest = Bad User
    log file = /var/log/samba/%m.log
    max log size = 50

[nginx_html]
    path = /usr/share/nginx/html
```

```
    writable = yes
    guest ok = yes
    guest only = yes
    force user = http
    create mask = 0664
    directory mask = 0775
```



Файл изменен. Нажмите последовательно:CTRL + O, затем клавишу Enter (для записи файла).CTRL + X (для выхода из редактора nano)

8.3.2.1. Проверка синтаксиса

выполните встроенную команду Samba для проверки синтаксиса файла конфигурации:

#bash

```
testparm -s
```

Тест синтаксиса пройден успешно (Loaded services file OK). Ошибок в файле smb.conf нет. Сетевая папка nginx_html определена верно.

8.3.2.2. Запуск и автозагрузка

Следующий шаг — запуск и добавление службы Samba в автозагрузку.

#bash

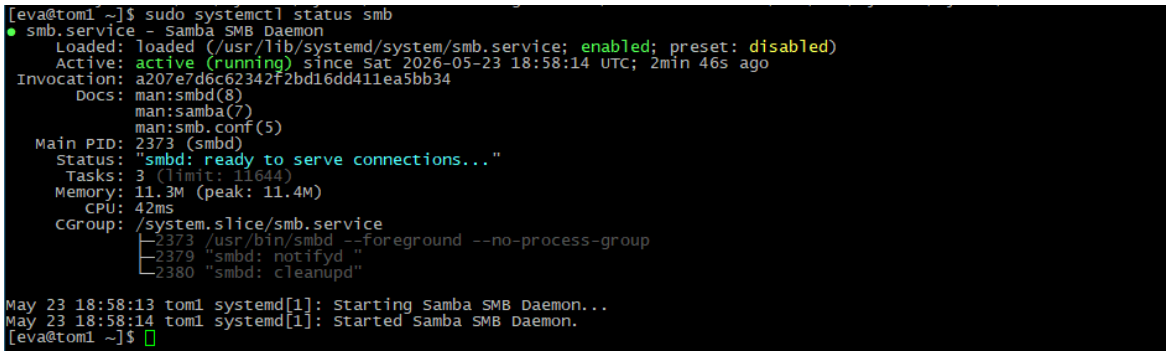
```
sudo systemctl enable --now smb
```

(Симлинк успешно создан, служба добавлена в автозапуск.)
Следующий шаг — обязательная проверка статуса запущенного демона Samba.

8.3.2.3. Проверка статуса

#bash

```
sudo systemctl status smb
```



(Служба smb работает в режиме active (running) и полностью готова принимать сетевые подключения.)
Службы настроены.

8.3.3. Владение корневым каталогом сайта

Передаем владение корневым каталогом сайта встроенному веб-пользователю http. Задаем права 775 для всех папок (чтобы Samba и Nginx могли создавать файлы) и 664 для файлов (только чтение и запись, без флагов исполнения):.
Выполните в терминале команду:

#bash

```
sudo chown -R http:http /usr/share/nginx/html/
sudo find /usr/share/nginx/html/ -type d -exec chmod 775 {} +
sudo find /usr/share/nginx/html/ -type f -exec chmod 664 {} +
```

```
[eva@tom1 ~]$ sudo chown -R http:http /usr/share/nginx/html/
[eva@tom1 ~]$ sudo find /usr/share/nginx/html/ -type d -exec chmod 775 {} +
[eva@tom1 ~]$ sudo find /usr/share/nginx/html/ -type f -exec chmod 664 {} +
[eva@tom1 ~]$
```

(Права 777/664 назначены успешно.)
Следующий обязательный шаг по нашему плану — проверка того, как система применила эти права к содержимому каталога.

#bash

```
ls -la /usr/share/nginx/html
```

```
drwxrwxr-x 1 http http  36 May 26 16:05 .
drwxr-xr-x 1 root root   8 May 21 16:06 ..
-rw-rw-r-- 1 http http 497 May 22 16:16 50x.html
-rw-rw-r-- 1 http http 896 May 22 16:16 index.html
[eva@tom1 ~]$
```

(Проверка прав прошла успешно. Строки drwxrwxr-x для текущей папки (.) и -rw-rw-r– для файлов index.html и 50x.html подтверждают, что доступ полностью открыт на чтение, запись для пользователей системы.)

8.3.3. Подключение сетевой папки в Windows

8.3.3.1. Получим точный текущий IP-адрес сервера tom_1

Выполните в терминале команду:

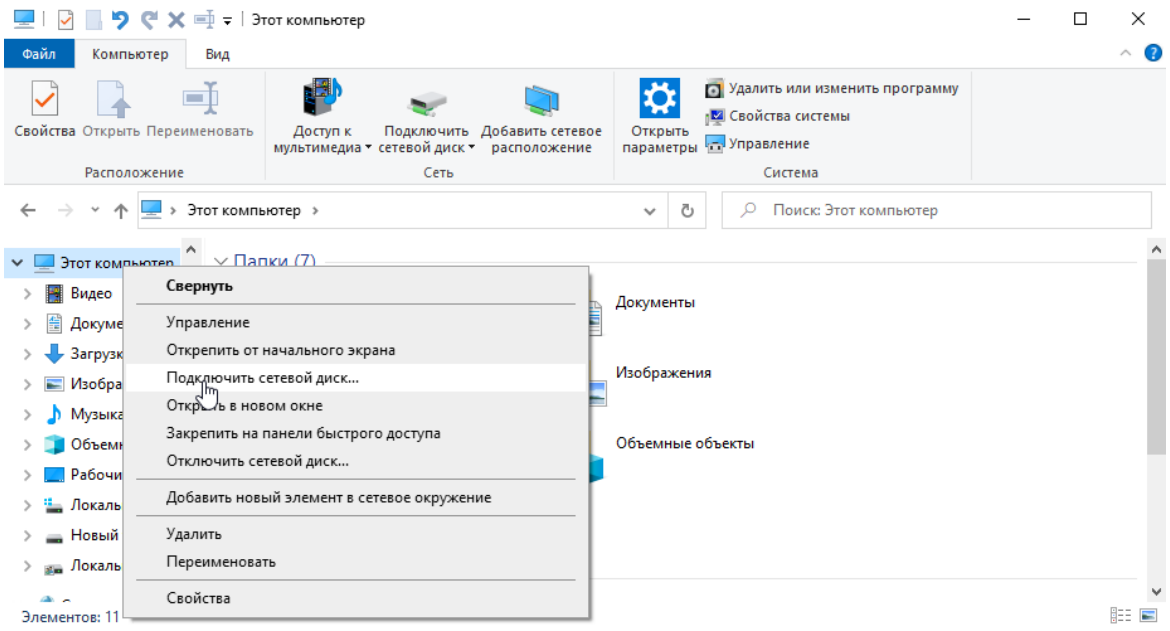
#bash

```
ip -br address show scope global | awk '{print $3}' | cut -d/ -f1
```

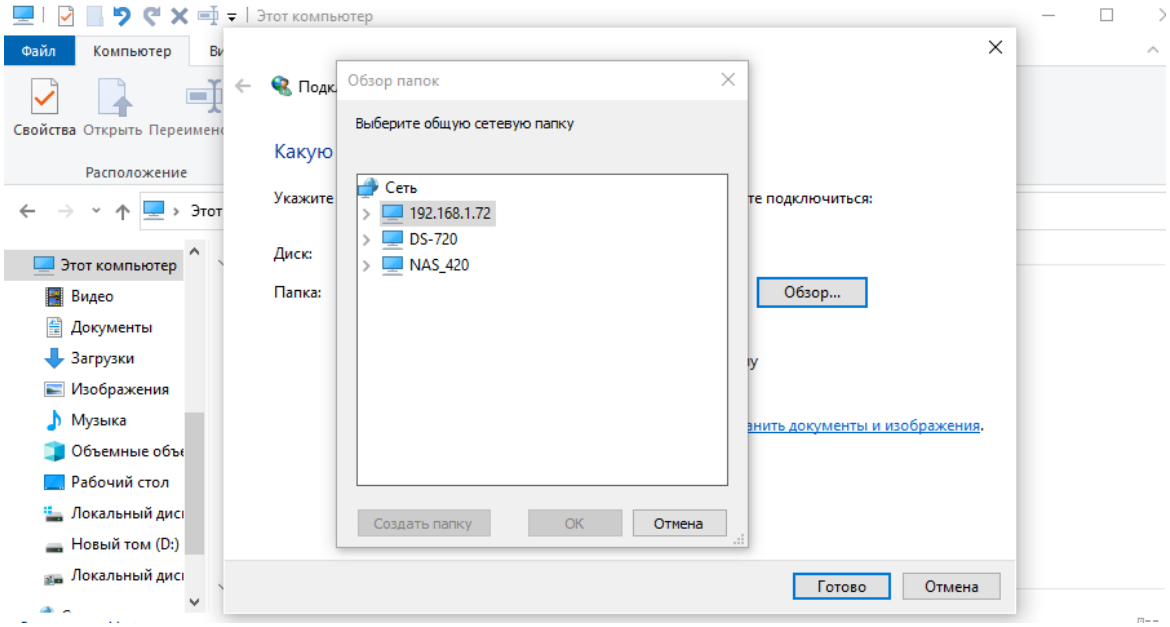
```
[eva@tom1 ~]$ ip -br address show scope global | awk '{print $3}' | cut -d/ -f1
192.168.1.72
[eva@tom1 ~]$
```

Теперь папка готова к подключению в качестве сетевого диска в среде Windows, чтобы вы могли открыть её через Notepad++.

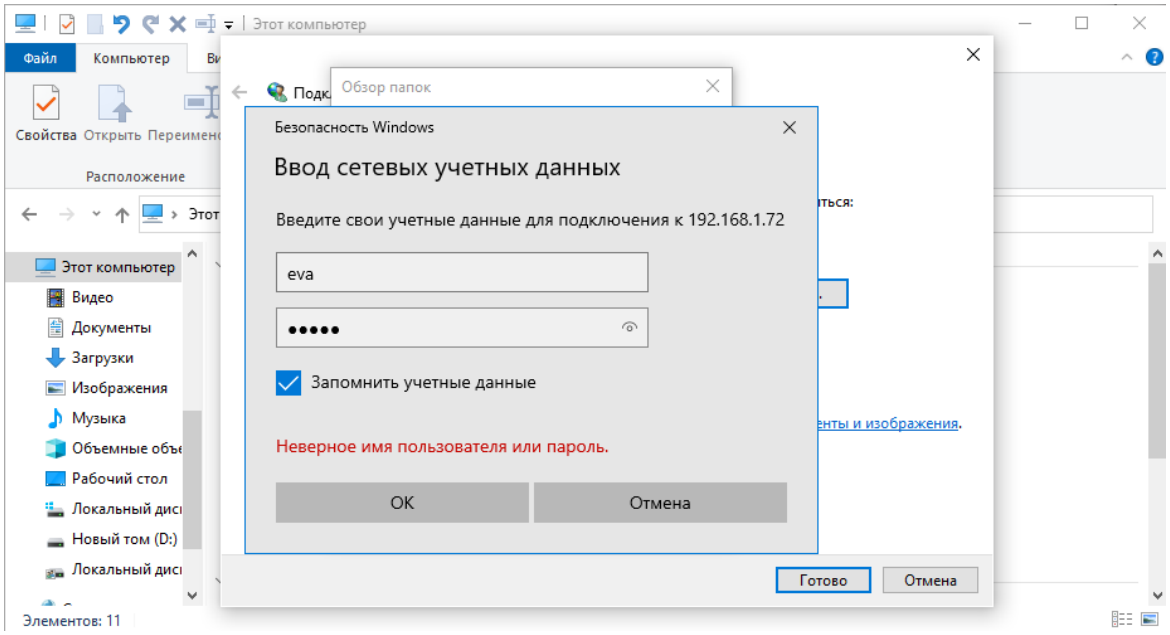
1. Откройте Проводник (Этот компьютер) на вашей Windows-машине.
2. В верхнем меню нажмите кнопку «Подключить сетевой диск» (или нажмите правой кнопкой мыши по «Этот компьютер» → «Подключить сетевой диск»).



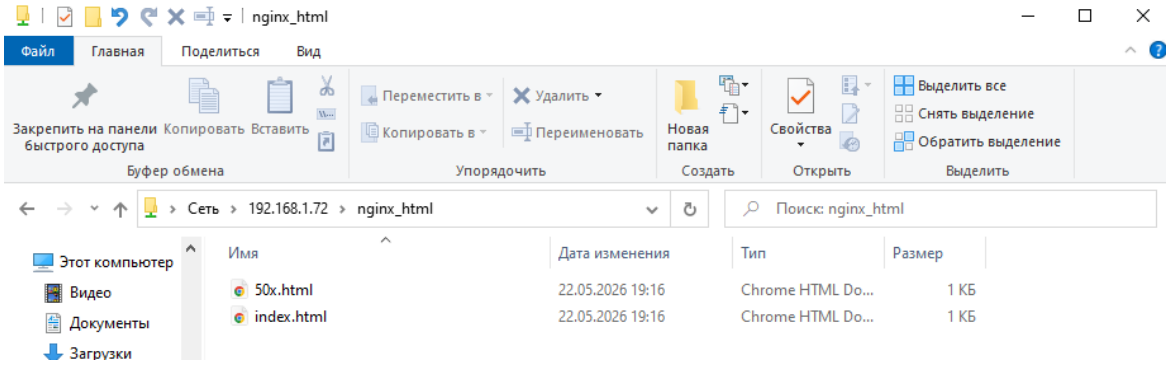
1. В поле «Папка» введите сетевой путь, используя IP-адрес вашего сервера tom_1 (из логов PuTTY: 192.168.1.72 или через кнопку обзор)



Введите сетевые учетные данные



Зайдите через проводник



(В корне /usr/share/nginx/html/ только файлы 50x.html и index.html.)
Разворачиваем структуру каталогов для нашего веб-интерфейса. Создадим стандартные папки для стилей, скриптов и серверной логики.

8.4. Системный пользователь http

Пользователь http в Arch Linux — это встроенный системный пользователь, от имени которого по умолчанию работают веб-серверы (например, Apache или Nginx) и сопутствующие им службы.
Он создается автоматически при установке этих программ для изоляции процессов и обеспечения безопасности.

Назначение

- Безопасность: Веб-серверы не должны работать под правами суперпользователя (root). Если злоумышленник найдет уязвимость в вашем сайте, он получит доступ только к файлам с правами пользователя http, что убережет остальную систему от взлома.

- Права на файлы: Этот пользователь владеет файлами и папками, к которым сервер имеет доступ (обычно они расположены в директории /srv/http/).
- Группа http: Для удобства существует одноименная группа http, в которую могут входить другие пользователи, чтобы иметь возможность редактировать файлы сайта без изменения прав доступа к ним через sudo

Применение

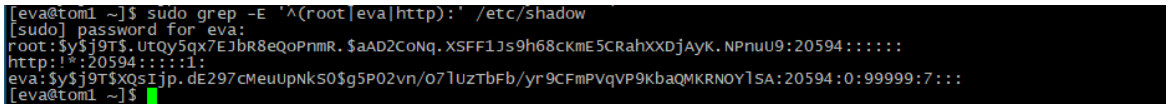
- Размещение сайтов: При настройке Nginx, Apache или PHP-FPM часто требуется указывать, что процесс должен запускаться от имени http.
- Настройка разрешений (Permissions): Если сайт выдает ошибку доступа (например, 403 Forbidden), обычно это означает, что у пользователя http нет прав на чтение нужных файлов или папок.
- Безопасность каталогов: Если скриптам на сайте нужно загружать файлы на сервер, папке загрузки необходимо выдать права (владельца) для пользователя http

Вывод строк пользователей системы

Выводим строки трех нами известных пользователей из базы данных.

#bash

```
sudo grep -E '^(root|eva|http):' /etc/shadow
```



root:yj9T$...:20594:::::

- Второе поле — длинный хэш y.... Это зашифрованный пароль суперпользователя.Число 20594 — дата последнего изменения пароля (в днях от 1970 года).В конце строки — пустые поля. Это значит, что для root нет никаких ограничений.

eva:yj9T$...:20594:0:99999:7:::

- Тоже видим хэш личного пароля.
- Параметры 0:99999:7 означают стандартные правила пользователя: пароль можно менять сразу (0), он действует 99999 дней, а за 7 дней до истечения система начнет предупреждать. Восьмое поле пустое — аккаунт не блокируется.

http:!*:20594::::::1:

- Второе поле содержит !*. Это маркер того, что пароль заблокирован (вход по паролю невозможен, учетка техническая).Внимание в самый конец строки: ::::::1:
- На предпоследней позиции (8-е поле) стоит цифра 1. В системе Linux это означает, что учетная запись принудительно заблокирована подсистемой безопасности PAM через 1 день после начала эпохи Unix (то есть 2 января 1970 года).

Изменяем параметры пользователя http

Уберем эту единицу из конца строки, чтобы сделать учетную запись бессрочной.

#bash

```
sudo chage -E -1 http
```

Проверка изменений в файле /etc/shadow

#bash

```
sudo grep '^http:' /etc/shadow
```

(Строка завершается чистыми двоеточиями (::::::), что означает: блокировка PAM полностью снята, аккаунт http стал бессрочным)

8.5. Настройка беспарольного доступа в sudoers

Чтобы PHP-скрипты могли вызывать утилиты управления аккаунтами без ввода пароля и без наличия текстового экрана (TTY), настроим правила безопасности. Команды будут пробрасываться во внешнюю систему через утилиту `systemd-run` для обхода ограничений безопасности PHP.

Откройте конфигурационный файл строго через visudo:

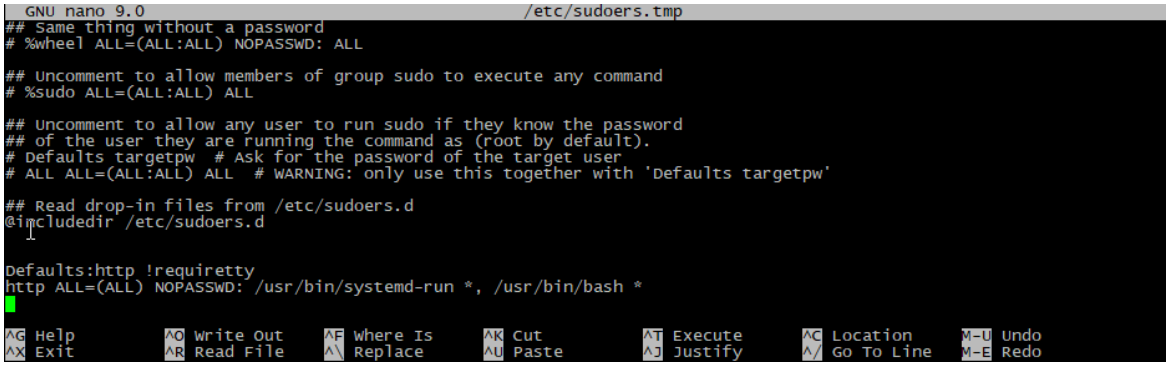
#bash

```
sudo EDITOR=nano visudo
```

В самый конец файла добавьте следующие строки:

text

```
Defaults:http !requiretty
http ALL=(ALL) NOPASSWD: /usr/bin/systemd-run *, /usr/bin/bash *
```



Обновление контекста PHP-FPM

Так как PHP кэширует права сессий для вызова exec(), обязательно примените изменения через перезапуск служб, поочередно введя 3 команды:

#bash

```
sudo systemctl daemon-reload
sudo systemctl restart php-fpm
sudo systemctl restart nginx
```

Структура web приложения

Папка веб-сервера nginx_html находится по пути /usr/share/nginx/html/ и имеет следующую структуру файлов бэкенда (PHP) и фронтенда (JS/CSS):

```
/usr/share/nginx/html/                  # Корневая директория веб-сервера Nginx
├── index.html                          # Главный интерфейс панели (вкладки,
таблицы, модальные окна)
├── css/
│   └── style.css                       # Стили оформления интерфейса панели
управления
├── js/
│   └── app.js                          # Клиентская логика (асинхронные Fetch-
запросы к API, фильтры)
└── api/
```

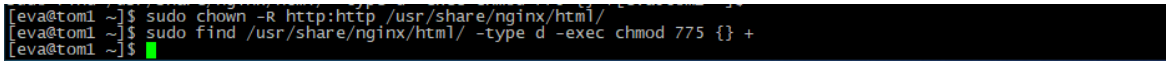
```
    ├── users.php                    # Серверный обработчик для системных
пользователей (/etc/passwd)
    └── groups.php                   # Серверный обработчик для системных групп
(/etc/group)
```

Создание директорий

Временно изменим права для работы в консоли

#bash

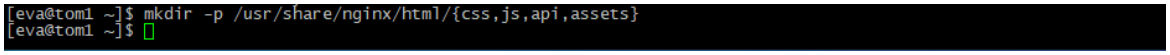
```
sudo chown -R http:http /usr/share/nginx/html/
sudo find /usr/share/nginx/html/ -type d -exec chmod 775 {} +
```



Выполните в терминале PuTTY одну команду:

#bash

```
mkdir -p /usr/share/nginx/html/{css,js,api,assets}
```

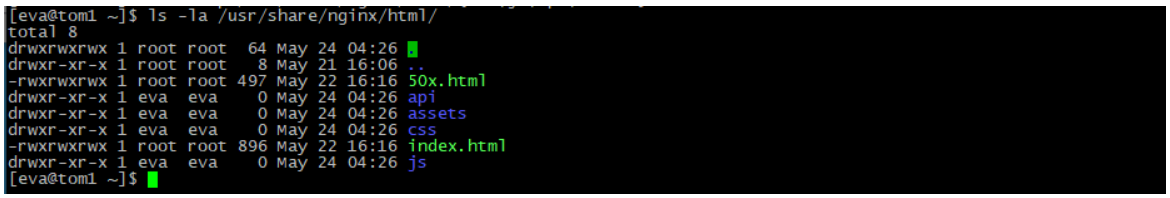


Папки созданы. Теперь обязательный шаг контроля: проверяем, какие права доступа и владельцы назначены для новых директорий, чтобы Windows-пользователь Samba и веб-сервер Nginx могли с ними работать.

Контроль папок

#bash

```
ls -la /usr/share/nginx/html/
```



(Папки создались под пользователем eva, но у них стоят ограниченные права drwxr-xr-x. Из-за этого Windows через Samba не сможет создавать или изменять файлы внутри этих подкаталогов.)

Права доступа к файлам

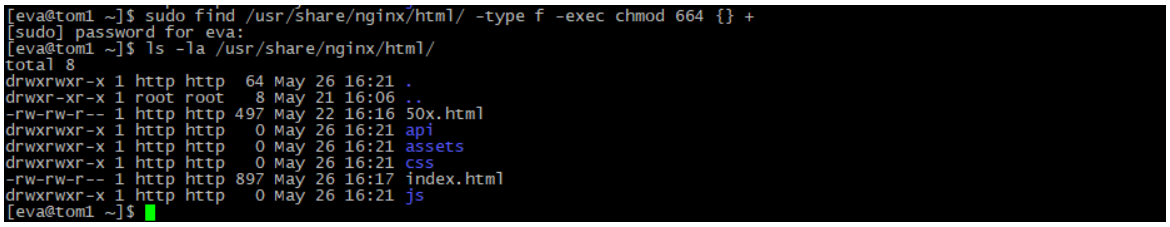
#bash

```
sudo find /usr/share/nginx/html/ -type f -exec chmod 664 {} +
```

Проверка назначения прав пользователя

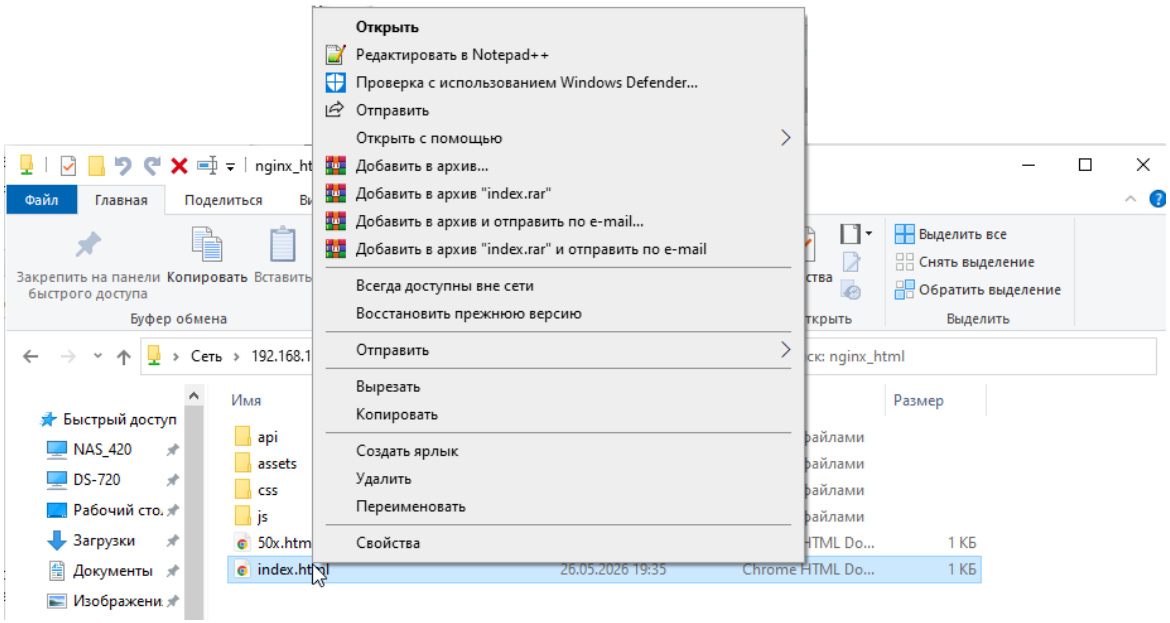
#bash

```
ls -la /usr/share/nginx/html/
```



(Права drwxrwxr-x (775) успешно применились ко всем новым директориям (api, assets, css, js) - подсвечены синим, и -rw-rw-r– к файлам они подсвечены белым. Теперь пользователи eva и системный пользователь http, и Samba имеют полный доступ.)

Прверим создание папок в Проводнике виндовс и откроем его в редакторе notepad++



9. Исходный код компонентов WebUI-инсталлятора

9.1. Главный интерфейс панели (index.html)

Сейчас мы с вами создадим тестовое приложение для нашего сервера, которое подтвердить правильность наших действий по настройке беспарольного доступа в sudoers и отключение системной изоляции PHP-FPM, а так же настройки сервера и прав на папки и файлы.
Сейчас мы не будем разбирать html, php и javascript тестового приложения, т.к. наша главная задача собрать iso - образ и при установке с флешки на сервер, убедиться в том, что все настройки сохранились и приложение взаимодействует с сервером, а написание всего web-приложения нас ждет позже, после тестирования iso-образа.
Отредактируйте файл index.html в редакторе. Целиком замените дефолтный код файла на приведенный ниже. Он формирует окно панели управления, вкладки переключения, таблицы и скрытые модальные формы:

index.html

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Control Panel</title>
    <link rel="stylesheet" href="css/style.css">
</head>
<body>
    <div class="window">
        <header class="window-header">
            <span class="title">Control Panel</span>
```

```
            <div class="window-controls">
                <button class="win-btn">?</button>
                <button class="win-btn">–</button>
                <button class="win-btn">□</button>
                <button class="win-btn close">×</button>
            </div>
        </header>

        <div class="window-body">
            <aside class="sidebar">
                <div class="search-box">
                    <input type="text" placeholder="⌕ Search">
                </div>
                <nav class="menu">
                    <div class="menu-group">^ File Sharing</div>
                    <a href="#" class="menu-item">▯ Shared Folder</a>
                    <a href="#" class="menu-item">⇆ File Services</a>
                    <a href="#" class="menu-item active">▯ User &
Group</a>
                    <a href="#" class="menu-item">▯ Domain/LDAP</a>
                    <div class="menu-group">^ Connectivity</div>
                    <a href="#" class="menu-item">▯ External Access</a>
                    <a href="#" class="menu-item">▯ Network</a>
                    <a href="#" class="menu-item">▯▯ Security</a>
                    <a href="#" class="menu-item">▯ Terminal & SNMP</a>
                </nav>
            </aside>

            <main class="main-content">
                <div class="tabs">
                    <button class="tab active" id="tab-
user">User</button>
                  <button class="tab" id="tab-group">Group</button>
                    <button class="tab">Advanced</button>
                </div>

                <div class="toolbar">
                    <div class="actions">
                        <button class="btn primary" id="btn-
create">Create</button>
                        <button class="btn" id="btn-edit"
disabled>Edit</button>
                        <button class="btn" id="btn-delete"
disabled>Delete</button>
                        <button class="btn">Export ▾</button>
                        <button class="btn">Delegate ▾</button>
                    </div>
                    <div class="filter">
                        <input type="text" id="table-filter"
placeholder="∇ Filter">
                    </div>
```

```
                </div>

                <div class="table-container">
                    <table id="users-table">
                        <thead>
                            <tr>
                                <th>Name ▲</th>
                                <th>Email</th>
                                <th>Description</th>
                                <th>2FA Status</th>
                                <th>Status</th>
                            </tr>
                        </thead>
                        <tbody>
             <!-- Данные загружаются через JS -->
                        </tbody>
                    </table>
                </div>

                <footer class="table-footer">
                    <span id="items-count">0 items</span>
                    <button id="refresh-btn" class="btn">↻</button>
                </footer>
            </main>
        </div>
    </div>

  <!-- Модальное окно Создания / Редактирования -->
    <div class="modal" id="user-modal">
        <div class="modal-content">
            <h3 id="modal-title">Create User</h3>
            <form id="user-form">
                <input type="hidden" id="form-action" value="create">
                <input type="hidden" id="old-username">

                <div class="form-group">
                    <label for="username">Имя пользователя:</label>
                    <input type="text" id="username" required
pattern="^[a-z_][a-z0-9_-]*$" title="Маленькие латинские буквы и цифры">
                </div>
                <div class="form-group">
                    <label for="description">Описание (GECOS):</label>
                    <input type="text" id="description">
                </div>
                <div class="form-group" id="password-group">
                    <label for="password">Пароль:</label>
                    <input type="password" id="password">
                </div>
                <div class="form-buttons">
                    <button type="button" class="btn" id="btn-modal-
cancel">Cancel</button>
```

```
                    <button type="submit" class="btn
primary">Save</button>
                </div>
            </form>
        </div>
    </div>

 <!-- Модальное окно Групп -->
  <div class="modal" id="group-modal">
      <div class="modal-content">
          <h3>Create Group</h3>
          <form id="group-form">
              <div class="form-group">
                  <label for="group-name">Имя группы:</label>
                  <input type="text" id="group-name" required
pattern="^[a-z_][a-z0-9_-]*$">
              </div>
              <div class="form-buttons">
                  <button type="button" class="btn" id="btn-group-
cancel">Cancel</button>
                  <button type="submit" class="btn
primary">Save</button>
              </div>
          </form>
      </div>
  </div>


    <script src="js/app.js"></script>
</body>
</html>
```

9.2. Стили оформления интерфейса (css/style.css)

Создайте файл css/style.css в подпапке css/. Код задает внешний вид окна приложения, таблиц данных, кнопок управления и всплывающих модальных окон:

style.css

```
*{
    box-sizing: border-box;
    margin: 0;
    padding: 0;
    font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto,
sans-serif;
}

body {
```

```
    background-color: #f0f2f5;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
}

.window {
    width: 1000px;
    height: 550px;
    background: #fff;
    border-radius: 6px;
    box-shadow: 0 5px 25px rgba(0,0,0,0.1);
    display: flex;
    flex-direction: column;
    overflow: hidden;
    border: 1px solid #dcdcdc;
}

.window-header {
    background: #fff;
    padding: 12px 15px;
    display: flex;
    justify-content: space-between;
    align-items: center;
    border-bottom: 1px solid #e2e8f0;
}

.window-header .title {
    font-size: 14px;
    color: #2d3748;
    font-weight: 500;
}

.window-controls .win-btn {
    border: none;
    background: none;
    padding: 4px 8px;
    cursor: pointer;
    color: #718096;
}

.window-body {
    display: flex;
    flex: 1;
    overflow: hidden;
}

/* Навигация */
.sidebar {
    width: 230px;
```

```
    background: #f7fafc;
    border-right: 1px solid #e2e8f0;
    padding: 12px;
}

.search-box input {
    width: 100%;
    padding: 6px 10px;
    border: 1px solid #cbd5e0;
    border-radius: 4px;
    margin-bottom: 15px;
}

.menu-group {
    font-size: 11px;
    text-transform: uppercase;
    color: #a0aec0;
    margin: 12px 0 6px 6px;
    font-weight: 600;
}

.menu-item {
    display: block;
    padding: 8px 12px;
    color: #4a5568;
    text-decoration: none;
    font-size: 13px;
    border-radius: 4px;
}

.menu-item.active {
    background: #ebf8ff;
    color: #2b6cb0;
    font-weight: 600;
}

/* Основная рабочая область */
.main-content {
    flex: 1;
    display: flex;
    flex-direction: column;
    padding: 0 20px;
}

.tabs {
    display: flex;
    border-bottom: 1px solid #e2e8f0;
    margin-top: 10px;
}

.tab {
```

```
    padding: 10px 20px;
    border: none;
    background: none;
    cursor: pointer;
    font-size: 14px;
    color: #718096;
}

.tab.active {
    color: #3182ce;
    border-bottom: 2px solid #3182ce;
    font-weight: 600;
}

.toolbar {
    display: flex;
    justify-content: space-between;
    margin: 15px 0;
}

.btn {
    padding: 6px 14px;
    border: 1px solid #cbd5e0;
    background: #fff;
    border-radius: 4px;
    cursor: pointer;
    font-size: 13px;
    color: #4a5568;
}

.btn:disabled {
    background: #f7fafc;
    color: #a0aec0;
    cursor: not-allowed;
    border-color: #e2e8f0;
}

.btn:hover:not(:disabled) {
    background: #f7fafc;
}

.btn.primary {
    background: #3182ce;
    color: #fff;
    border-color: #3182ce;
}

.btn.primary:hover {
    background: #2b6cb0;
}
```

```
.filter input {
    padding: 6px 10px;
    border: 1px solid #cbd5e0;
    border-radius: 4px;
    font-size: 13px;
}

/* Таблица */
.table-container {
    flex: 1;
    overflow-y: auto;
    border: 1px solid #e2e8f0;
    border-radius: 4px;
}

table {
    width: 100%;
    border-collapse: collapse;
    font-size: 13px;
}

th, td {
    padding: 10px 12px;
    text-align: left;
    border-bottom: 1px solid #edf2f7;
    user-select: none;
}

th {
    background: #f7fafc;
    color: #4a5568;
    font-weight: 600;
    position: sticky;
    top: 0;
}

tbody tr {
    cursor: pointer;
}

tbody tr:hover {
    background: #f7fafc;
}

tr.selected-user {
    background: #e8f0fe !important;
}

.status-deactivated { color: #e53e3e; }
.status-normal { color: #38a169; }
```

```
.table-footer {
    display: flex;
    justify-content: flex-end;
    align-items: center;
    padding: 12px 0;
    gap: 15px;
    font-size: 13px;
    color: #718096;
}

/* Модальные окна */
.modal {
    display: none;
    position: fixed;
    top: 0; left: 0; width: 100%; height: 100%;
    background: rgba(0,0,0,0.4);
    justify-content: center;
    align-items: center;
    z-index: 1000;
}

.modal.open {
    display: flex;
}

.modal-content {
    background: #fff;
    padding: 20px;
    border-radius: 6px;
    width: 400px;
    box-shadow: 0 4px 15px rgba(0,0,0,0.2);
}

.modal-content h3 {
    margin-bottom: 15px;
    color: #2d3748;
}

.form-group {
    margin-bottom: 12px;
}

.form-group label {
    display: block;
    font-size: 12px;
    color: #4a5568;
    margin-bottom: 4px;
}

.form-group input {
    width: 100%;
```

```
    padding: 8px;
    border: 1px solid #cbd5e0;
    border-radius: 4px;
}

.form-buttons {
    display: flex;
    justify-content: flex-end;
    gap: 10px;
    margin-top: 18px;
}
```

9.3. Серверный обработчик пользователей (api/users.php)

Создайте файл `api/users.php` в подпапке `api/`. Скрипт обрабатывает GET-запросы для вывода учетных записей (исключая технического nobody) и POST-запросы для выполнения атомарных операций через `systemd-run` в обход изоляции:

users.php

```
<?php
header('Content-Type: application/json; charset=utf-8');

// Обработка получения списка (GET)
if ($_SERVER['REQUEST_METHOD'] === 'GET') {
    $usersList = [];

    if (is_readable('/etc/passwd')) {
        $lines = file('/etc/passwd', FILE_IGNORE_NEW_LINES |
FILE_SKIP_EMPTY_LINES);
        foreach ($lines as $line) {
            $parts = explode(':', $line);
            if (count($parts) >= 5) {
                $username = $parts[0];
                $uid = (int)$parts[2];
                $description = $parts[4];

                // Выводим root (UID 0), системных и обычных пользователей
(UID >= 1000)
                if (($uid === 0 || $uid >= 1000) && $username !==
'nobody' || $username === 'guest' || $username === 'admin') {


                    // Сопоставление статуса активности учетной записи
                    $status = 'Normal';
                    if ($username === 'guest') {
                        $status = 'Deactivated';
                    }
```

```
                    $usersList[] = [
                        'name' => $username,
                        'email' => '',
                        'desc' => $description,
                        'tfa' => 'Disabled',
                        'status' => $status
                    ];
                }
            }
        }
    }
    echo json_encode($usersList, JSON_UNESCAPED_UNICODE);
    exit;
}

// Обработка действий создания/изменения/удаления (POST)
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $input = json_decode(file_get_contents('php://input'), true);

    if (!isset($input['action'])) {
        echo json_encode(['success' => false, 'error' => 'Отсутствует
действие']);
        exit;
    }

    $action = $input['action'];
    $username = preg_replace('/[^a-z0-9_-]/', '', $input['username'] ??
'');
    $description = escapeshellarg($input['description'] ?? '');
    $password = $input['password'] ?? '';

    if (empty($username)) {
        echo json_encode(['success' => false, 'error' => 'Некорректное
имя пользователя']);
        exit;
    }

    switch ($action) {
        case 'create':
            if (empty($username)) {
                echo json_encode(['success' => false, 'error' =>
'Некорректное имя пользователя']);
                exit;
            }

            // Генерируем хэш пароля
            $salt = '$1$' . substr(md5(uniqid(rand(), true)), 0, 8) .
'$';
            $hashed_password = crypt($password, $salt);
            $uid_gid = rand(1100, 1900);
```

```
            $passwd_line =
"{$username}:x:{$uid_gid}:{$uid_gid}:{$description}:/home/{$username}:/
bin/bash";
            $shadow_line =
"{$username}:{$hashed_password}:19500:0:99999:7:::";
            $group_line = "{$username}:x:{$uid_gid}:";

            // Собираем все команды в одну строку для запуска через bash
            $system_cmd = "echo '{$passwd_line}' >> /etc/passwd && " .
                          "echo '{$shadow_line}' >> /etc/shadow && " .
                          "echo '{$group_line}' >> /etc/group && " .
                          "mkdir -p /home/{$username} && " .
                          "chown -R {$uid_gid}:{$uid_gid}
/home/{$username}";

            // Вызываем встроенную службу systemd-run. Флаг -G заставляет её
отработать от root наружу.
            $cmd = "sudo /usr/bin/systemd-run -G /usr/bin/bash -c " .
escapeshellarg($system_cmd) . " 2>&1";

            exec($cmd, $output, $return_var);

            if ($return_var !== 0) {
                $err = implode(' ', $output);
                echo json_encode(['success' => false, 'error' =>
"Ошибка запуска: {$err} (Код: {$return_var})"]);
                exit;
            }

            echo json_encode(['success' => true]);
            break;


        case 'update':
            // Читаем параметры из JSON-запроса от браузера
            $old_username = preg_replace('/[^a-z0-9_-]/', '',
$input['old_username'] ?? '');
            $new_description = $input['description'] ?? '';
            $new_password = $input['password'] ?? '';

            if (empty($old_username)) {
                echo json_encode(['success' => false, 'error' => 'Не
указан пользователь для редактирования']);
                exit;
            }

            // Экранируем описание, чтобы оно не сломало синтаксис команды sed
            $clean_desc = str_replace('/', '\/', $new_description);

            // 1. Команда для обновления описания (GECOS) в /etc/passwd
```

```
            $update_cmd = "sed -i -E
's/^({$old_username}:[^:]*:[^:]*:[^:]*):[^:]*(:.*)/\\1:{$clean_desc}\\2
/' /etc/passwd";

            // 2. Если в форму ввели новый пароль — добавляем команду обновления
хэша в /etc/shadow
            if (!empty($new_password)) {
                $salt = '$1$' . substr(md5(uniqid(rand(), true)), 0, 8)
. '$';
                $hashed_password = crypt($new_password, $salt);
                $clean_hash = str_replace('/', '\/', $hashed_password);

                $update_cmd .= " && sed -i -E
's/^({$old_username}:)[^:]*(:.*)/\\1{$clean_hash}\\2/' /etc/shadow";
            }

            // Вызываем итоговую команду через systemd-run наружу от root
            $cmd = "sudo /usr/bin/systemd-run -G /usr/bin/bash -c " .
escapeshellarg($update_cmd) . " 2>&1";

            exec($cmd, $output, $return_var);

            if ($return_var === 0) {
                echo json_encode(['success' => true]);
            } else {
                $err = implode(' ', $output);
                echo json_encode(['success' => false, 'error' =>
"Ошибка обновления: {$err}"]);
            }
            break;



        case 'delete':
            if ($username === 'root') {
                echo json_encode(['success' => false, 'error' =>
'Удаление root запрещено']);
                exit;
            }

            // Формируем команды для полной очистки записей из passwd,
shadow, group и удаления папки
            $delete_cmd = "sed -i '/^{$username}:/d' /etc/passwd && " .
                          "sed -i '/^{$username}:/d' /etc/shadow && " .
                          "sed -i '/^{$username}:/d' /etc/group && " .
                          "rm -rf /home/{$username}";

            // Вызываем команду через systemd-run наружу от имени root
            $cmd = "sudo /usr/bin/systemd-run -G /usr/bin/bash -c " .
escapeshellarg($delete_cmd) . " 2>&1";
```

```
            exec($cmd, $output, $return_var);

            if ($return_var === 0) {
                echo json_encode(['success' => true]);
            } else {
                $err = implode(' ', $output);
                echo json_encode(['success' => false, 'error' =>
"Ошибка удаления: {$err}"]);
            }
            break;


        default:
            echo json_encode(['success' => false, 'error' => 'Неизвестная
операция']);
            break;
    }
    exit;
}
```

9.4. Серверный обработчик групп (api/groups.php)

Создайте файл `api/groups.php` в подпапке `api/`. Скрипт парсит системный файл `/etc/group`, выстраивает связи участников и нативно создаёт/удаляет группы в ОС через `systemd-run`:

groups.php

```
<?php
header('Content-Type: application/json; charset=utf-8');

$input = json_decode(file_get_contents('php://input'), true);
$action = $input['action'] ?? $_SERVER['REQUEST_METHOD'];

// --- ОБРАБОТКА ПОЛУЧЕНИЯ СПИСКА ГРУПП (GET) ---
if ($action === 'GET') {
    $groupsList = [];

    if (is_readable('/etc/group')) {
        $lines = file('/etc/group', FILE_IGNORE_NEW_LINES |
FILE_SKIP_EMPTY_LINES);
        foreach ($lines as $line) {
            // Формат /etc/group: имя_группы:пароль:GID:список_пользователей
            $parts = explode(':', $line);
            if (count($parts) >= 3) {
                $group_name = $parts[0];
                $gid = (int)$parts[2];
                $users = $parts[3] ?? ''; // Пользователи через запятую
```

```
                // Фильтруем системные группы, оставляем root (GID 0), wheel
и кастомные (GID >= 1000)
                if ($gid === 0 || $gid === 998 || $gid >= 1000) {
                    // Исключаем технического nobody
                    if ($group_name !== 'nobody') {
                        $groupsList[] = [
                            'name' => $group_name,
                            'gid' => $gid,
                            'users' => empty($users) ? '—' :
str_replace(',', ', ', $users),
                            'status' => 'Normal'
                        ];
                    }
                }
            }
        }
    }
    echo json_encode($groupsList, JSON_UNESCAPED_UNICODE);
    exit;
}

// --- ОБРАБОТКА ИЗМЕНЕНИЙ (POST) ---
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $group_name = preg_replace('/[^a-z0-9_-]/', '',
$input['group_name'] ?? '');

    if (empty($group_name)) {
        echo json_encode(['success' => false, 'error' => 'Некорректное
имя группы']);
        exit;
    }

    switch ($input['action']) {
        case 'create':
            // Создание группы через systemd-run наружу от root
            $cmd = "sudo /usr/bin/systemd-run -G /usr/bin/bash -c " .
escapeshellarg("groupadd {$group_name}") . " 2>&1";
            exec($cmd, $output, $return_var);

            if ($return_var === 0) {
                echo json_encode(['success' => true]);
            } else {
                $err = implode(' ', $output);
                echo json_encode(['success' => false, 'error' =>
"Ошибка создания группы: {$err}"]);
            }
            break;

        case 'delete':
            if ($group_name === 'root' || $group_name === 'wheel') {
```

```
                echo json_encode(['success' => false, 'error' =>
'Удаление системных групп запрещено']);
                exit;
            }
            // Удаление группы
            $cmd = "sudo /usr/bin/systemd-run -G /usr/bin/bash -c " .
escapeshellarg("groupdel {$group_name}") . " 2>&1";
            exec($cmd, $output, $return_var);

            if ($return_var === 0) {
                echo json_encode(['success' => true]);
            } else {
                $err = implode(' ', $output);
                echo json_encode(['success' => false, 'error' =>
"Ошибка удаления группы: {$err}"]);
            }
            break;

        default:
            echo json_encode(['success' => false, 'error' => 'Неизвестная
операция']);
            break;
    }
    exit;
}
```

9.5. Клиентские скрипты логики (js/app.js)

Создайте файл `js/app.js` в подпапке `js/`. Скрипт управляет асинхронным обновлением таблиц (fetch), переключением контекста вкладок, фильтрацией на лету и валидацией полей ввода:

app.js

```
document.addEventListener('DOMContentLoaded', () => {
    const tableBody = document.querySelector('#users-table tbody');
    const itemsCount = document.getElementById('items-count');
    const refreshBtn = document.getElementById('refresh-btn');
    const filterInput = document.getElementById('table-filter');

    // Кнопки управления
    const btnCreate = document.getElementById('btn-create');
    const btnEdit = document.getElementById('btn-edit');
    const btnDelete = document.getElementById('btn-delete');

    // Элементы модального окна
    const userModal = document.getElementById('user-modal');
    const userForm = document.getElementById('user-form');
```

```
    const modalTitle = document.getElementById('modal-title');
    const btnModalCancel = document.getElementById('btn-modal-cancel');

    let selectedUsername = null;
    let selectedUserRow = null;

    // Загрузка данных
    async function loadUsers() {
        try {
            const response = await fetch('api/users.php');
            if (!response.ok) throw new Error('Ошибка сервера');
            const data = await response.json();
            renderTable(data);
            resetSelection();
        } catch (error) {
            alert('Не удалось обновить список пользователей');
        }
    }

    function renderTable(users) {
        tableBody.innerHTML = '';
        users.forEach(user => {
            const tr = document.createElement('tr');
            tr.dataset.username = user.name;
            tr.dataset.desc = user.desc;

            const statusClass = user.status === 'Normal' ? 'status-
normal' : 'status-deactivated';

            tr.innerHTML = `
                <td><b>${escapeHtml(user.name)}</b></td>
                <td>${escapeHtml(user.email)}</td>
                <td>${escapeHtml(user.desc)}</td>
                <td>${escapeHtml(user.tfa)}</td>
                <td
class="${statusClass}">${escapeHtml(user.status)}</td>
            `;

            // Логика выбора строки кликом
            tr.addEventListener('click', () => {
                if (selectedUserRow)
selectedUserRow.classList.remove('selected-user');

                if (selectedUsername === user.name) {
                    resetSelection();
                } else {
                    selectedUsername = user.name;
                    selectedUserRow = tr;
                    tr.classList.add('selected-user');
                    btnEdit.disabled = false;
                    // Запрещаем удалять root-пользователя напрямую из UI
```

```
ради безопасности
                    btnDelete.disabled = (user.name === 'root');
                }
            });

            tableBody.appendChild(tr);
        });
        itemsCount.textContent = `${users.length} items`;
    }

    function resetSelection() {
        selectedUsername = null;
        selectedUserRow = null;
        btnEdit.disabled = true;
        btnDelete.disabled = true;
    }

    // Фильтрация таблицы
    filterInput.addEventListener('input', (e) => {
        const value = e.target.value.toLowerCase();
        Array.from(tableBody.querySelectorAll('tr')).forEach(tr => {
            const match = tr.textContent.toLowerCase().includes(value);
            tr.style.display = match ? '' : 'none';
        });
    });

    // Открытие модального окна создания
    btnCreate.addEventListener('click', () => {
        userForm.reset();
        document.getElementById('form-action').value = 'create';
        document.getElementById('username').disabled = false;
        document.getElementById('password-group').style.display =
'block';
        modalTitle.textContent = 'Create User';
        userModal.classList.add('open');
    });

    // Открытие модального окна редактирования
    btnEdit.addEventListener('click', () => {
        if (!selectedUsername) return;
        userForm.reset();
        document.getElementById('form-action').value = 'update';
        document.getElementById('old-username').value =
selectedUsername;

        const usernameInput = document.getElementById('username');
        usernameInput.value = selectedUsername;
        usernameInput.disabled = true; // Имя пользователя в Linux менять
через useradd напрямую нельзя

        document.getElementById('description').value =
```

```
selectedUserRow.dataset.desc || '';
        document.getElementById('password-group').style.display =
'block'; // Пароль заполнять по желанию

        modalTitle.textContent = 'Edit User';
        userModal.classList.add('open');
    });

    // Обработка кнопки удаления
    btnDelete.addEventListener('click', async () => {
        if (!selectedUsername) return;
        if (confirm(`Вы уверены, что хотите удалить пользователя
${selectedUsername} вместе с домашней директорией?`)) {
            await sendAction({ action: 'delete', username:
selectedUsername });
        }
    });

    // Закрытие модального окна
    btnModalCancel.addEventListener('click', () =>
userModal.classList.remove('open'));

    // Отправка формы (Создание / Изменение)
    userForm.addEventListener('submit', async (e) => {
        e.preventDefault();
        const action = document.getElementById('form-action').value;

        const payload = {
            action: action,
            username: document.getElementById('username').value,
            description: document.getElementById('description').value,
            password: document.getElementById('password').value,
            old_username: document.getElementById('old-username').value
        };

        await sendAction(payload);
        userModal.classList.remove('open');
    });

    async function sendAction(data) {
        try {
            const response = await fetch('api/users.php', {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify(data)
            });
            const result = await response.json();
            if (result.success) {
                loadUsers();
            } else {
                alert('Ошибка:' + result.error);
```

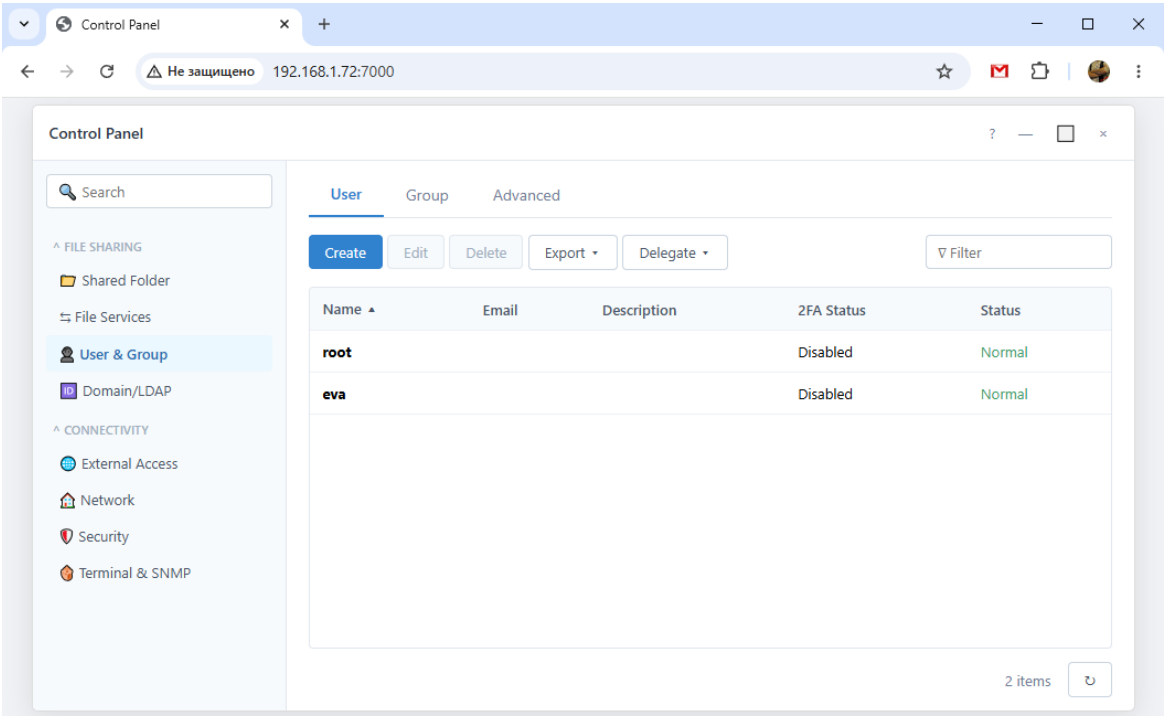
```
            }
        } catch (error) {
            alert('Ошибка сети при отправке запроса');
        }
    }

    function escapeHtml(text) {
        if (!text) return '';
        return text.toString().replace(/&/g, "&amp;").replace(/</g,
"&lt;").replace(/>/g, "&gt;");
    }

    refreshBtn.addEventListener('click', loadUsers);
    loadUsers();

    // --- ЛОГИКА ВКЛАДКИ GROUP ---
    const tabUser = document.getElementById('tab-user');
    const tabGroup = document.getElementById('tab-group');
    const tableHeader = document.querySelector('#users-table thead
tr');

    const groupModal = document.getElementById('group-modal');
    const groupForm = document.getElementById('group-form');
    const btnGroupCancel = document.getElementById('btn-group-cancel');

    let currentTab = 'user'; // Храним активную вкладку ('user' или
'group')
    let selectedGroupName = null;

    // Переключение на вкладку User
    tabUser.addEventListener('click', () => {
        tabGroup.classList.remove('active');
        tabUser.classList.add('active');
        currentTab = 'user';
        tableHeader.innerHTML = `
            <th>Name ▲</th>
            <th>Email</th>
            <th>Description</th>
            <th>2FA Status</th>
            <th>Status</th>
        `;
        resetSelection();
        loadUsers(); // Вызываем старую функцию загрузки пользователей
    });

    // Переключение на вкладку Group
    tabGroup.addEventListener('click', () => {
        tabUser.classList.remove('active');
        tabGroup.classList.add('active');
        currentTab = 'group';
        tableHeader.innerHTML = `
```

```
            <th>Group Name ▲</th>
            <th>GID</th>
            <th>Members (Users)</th>
            <th>Status</th>
        `;
        resetSelection();
        loadGroups();
    });

    // Загрузка групп с бэкенда
    async function loadGroups() {
        try {
            const response = await fetch('api/groups.php');
            const groups = await response.json();
            renderGroupsTable(groups);
        } catch (error) {
            alert('Ошибка загрузки групп');
        }
    }

    function renderGroupsTable(groups) {
        tableBody.innerHTML = '';
        groups.forEach(group => {
            const tr = document.createElement('tr');
            tr.innerHTML = `
                <td><b>${escapeHtml(group.name)}</b></td>
                <td>${group.gid}</td>
                <td>${escapeHtml(group.users)}</td>
                <td class="status-normal">${group.status}</td>
            `;

            tr.addEventListener('click', () => {
                if (selectedUserRow)
selectedUserRow.classList.remove('selected-user');

                if (selectedGroupName === group.name) {
                    selectedGroupName = null;
                    selectedUserRow = null;
                    btnDelete.disabled = true;
                } else {
                    selectedGroupName = group.name;
                    selectedUserRow = tr;
                    tr.classList.add('selected-user');
                    btnEdit.disabled = true; // Для групп редактирование
отключим
                    btnDelete.disabled = (group.name === 'root' ||
group.name === 'wheel');
                }
            });
            tableBody.appendChild(tr);
        });
```

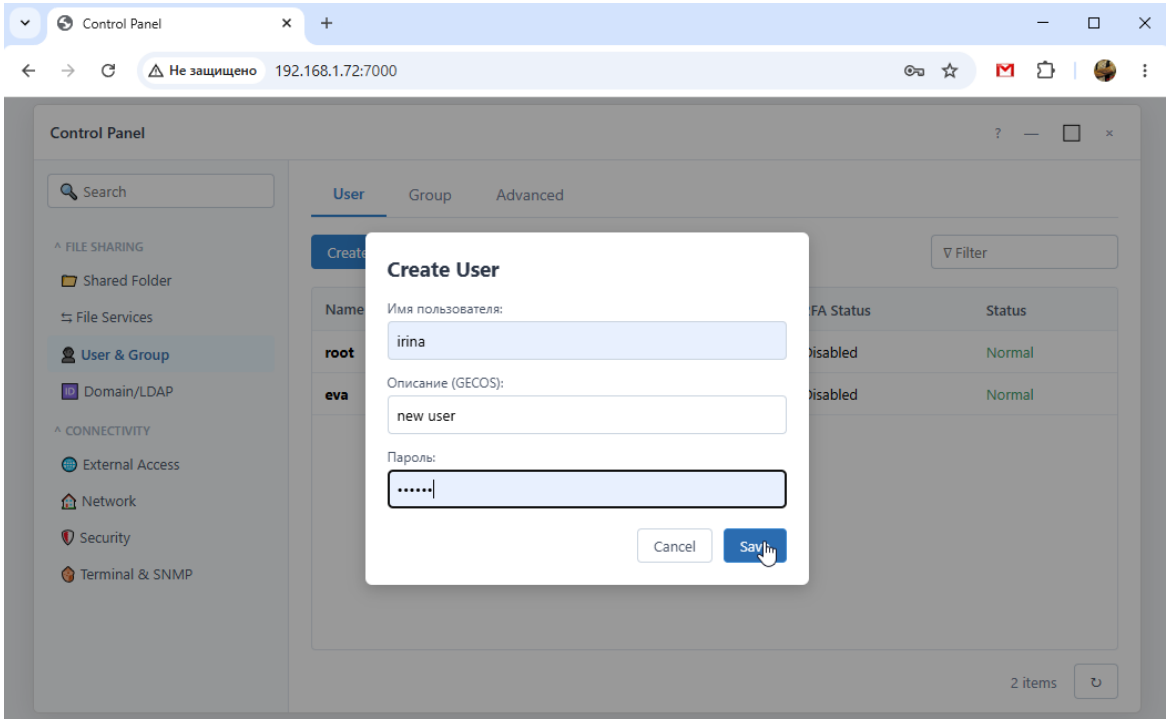
```
        itemsCount.textContent = `${groups.length} items`;
    }

    // Модифицируем общие кнопки под контекст активной вкладки
    btnCreate.addEventListener('click', (e) => {
        if (currentTab === 'group') {
            e.stopPropagation(); // Останавливаем открытие модалки
пользователей
            groupForm.reset();
            groupModal.classList.add('open');
        }
    });

    btnDelete.addEventListener('click', async () => {
        if (currentTab === 'group' && selectedGroupName) {
            if (confirm(`Удалить группу ${selectedGroupName}?`)) {
                await sendGroupAction({ action: 'delete', group_name:
selectedGroupName });
            }
        }
    });

    btnGroupCancel.addEventListener('click', () =>
groupModal.classList.remove('open'));

    groupForm.addEventListener('submit', async (e) => {
        e.preventDefault();
        await sendGroupAction({
            action: 'create',
            group_name: document.getElementById('group-name').value
        });
        groupModal.classList.remove('open');
    });

    async function sendGroupAction(data) {
        try {
            const response = await fetch('api/groups.php', {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify(data)
            });
            const result = await response.json();
            if (result.success) {
                loadGroups();
            } else {
                alert('Ошибка:' + result.error);
            }
        } catch (error) {
            alert('Ошибка сети при обработке группы');
        }
    }
```

```
    // Модифицируем круглую кнопку обновления (↻)
    refreshBtn.addEventListener('click', () => {
        if (currentTab === 'group') loadGroups();
    });


});
```

Сохраним файлы и проверим в окне браузера, перейдя по ссылке http://192.168.1.72:7000/



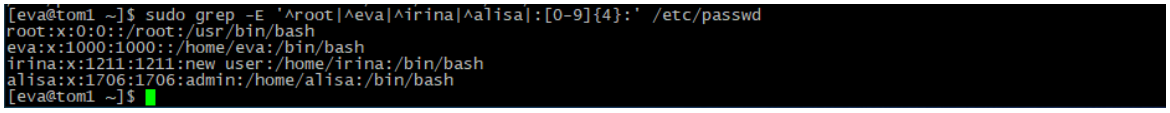
мы видим вывод в таблице наших пользователей root и eva, проверим работу web-страницы на предмет добавления пользователя irina с описанием new user и alisa / admin



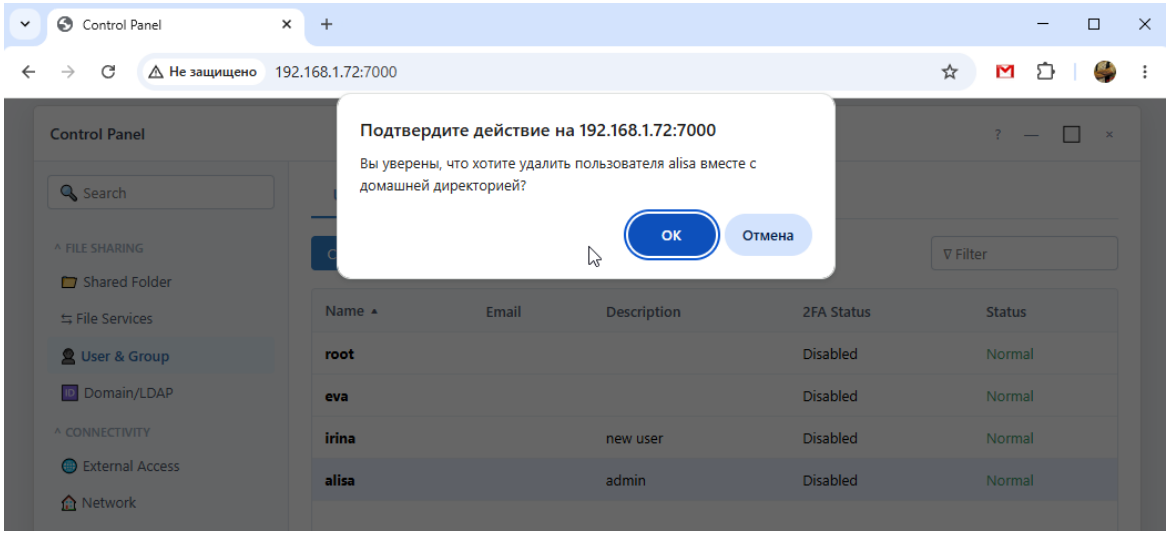
Проверим новых пользователей в консоли

#bash

```
sudo grep -E '^root|^eva|^irina|^alisa|:[0-9]{4}:' /etc/passwd
```



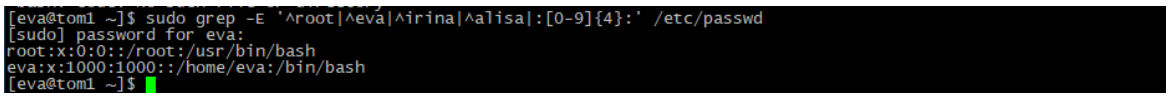
Внимание! Пользователей root и eva не удаляем, иначе консоль отключиться и мы с вами больше не попадем в управление сервером, а без пользователей с правами суперпользователей система нас просто не пустит.
Удалим строго только новых пользователей в веб-приложении.



И снова проверим консоль

#bash

```
sudo grep -E '^root|^eva|^irina|^alisa|:[0-9]{4}:' /etc/passwd
```



- Скачать файлы тестового web-приложения

!!!!!!!!!!! Пункт в Разработке !!!!!!!!!!!

Создание директорий

Не забыть диск Recovery !!!!!!!!!!!!

0. Создание Arch Linux Recovery

Для создания изолированного раздела восстановления в Arch Linux проще всего использовать связку из отдельного раздела на диске, образа Arch Linux ISO и загрузчика GRUB. Это позволит загружаться в полноценную среду восстановления прямо с диска, даже если основная система перестала работать.Ниже приведена пошаговая инструкция для реализации этого решения.

0.1. Подготовка раздела

0.1.1.Откройте терминал и запустите утилиту для разметки диска fdisk , указав нужный диск:

#bash

```
sudo fdisk /dev/nvme0n1 (или /dev/sda)
```

0.1.2. Создайте новый раздел размером от 1.5 до 3 ГБ (достаточный для размещения ISO-образа).

0.1.3. Измените его тип (код) на стандартный раздел данных Linux.

0.1.4. Отформатируйте раздел в файловую систему ext4:

#bash

```
sudo mkfs.ext4 /dev/nvme0n1pX (укажите номер вашего раздела)
```

0.2. Загрузка и перенос ISO-образа

0.2.1. Смонтируйте созданный раздел:

#bash

```
sudo mount /dev/nvme0n1pX /mnt
```

=== 0.2.2. Перейдите на Arch Linux Downloads и скопируйте прямую ссылку на актуальный archlinux-x86_64.iso (или используйте wget).

0.2.3. Скачайте образ прямо в корень смонтированного раздела:

#bash

```
sudo wget -O /mnt/arch-recovery.iso ССЫЛКА_НА_ОФИЦИАЛЬНЫЙ_ISO
```

0.3. Настройка загрузчика (GRUB)

Для того чтобы GRUB видел образ и мог в него загрузиться, нужно добавить специальный пункт в конфигурацию.

0.3.1. Установите пакет grub-imageboot, если он есть в репозитории (или добавьте параметры вручную).

0.3.2. Отредактируйте файл /etc/grub.d/40_custom:

#bash

```
sudo nano /etc/grub.d/40_custom
```

0.3.3. Добавьте следующий блок конфигурации в конец файла (замените UUID на UUID вашего раздела с ISO-образом, узнать его можно командой lsblk -f):

#bash

```
menuentry "Arch Linux Recovery" {
insmod ext2
insmod loopback
insmod iso9660
# Указываем UUID раздела восстановления
search --no-floppy --fs-uuid --set=root ВАШ_UUID_РАЗДЕЛА
loopback loop /arch-recovery.iso
linux (loop)/arch/boot/x86_64/vmlinuz-linux img_dev=/dev/disk/by-uuid/
ВАШ_UUID_РАЗДЕЛА img_loop=/arch-recovery.iso
archisobasedir=arch cow_space_size=256M earlymodules=loop
initrd (loop)/arch/boot/x86_64/initramfs-linux.img
}
```

0.4. Обновите конфигурацию GRUB, чтобы изменения вступили в силу:

sudo grub-mkconfig -o /boot/grub/grub.cfg </code> Теперь при перезагрузке компьютера вы сможете выбрать пункт «Arch Linux Recovery» в меню GRUB, который запустит Live-окружение для восстановления основной системы.

Пункт в разработке !!!!!!

20. Сокращение времени выбора загрузки

20.1 Сокращение времени выбора systemd-boot

Чтобы сократить время выбора операционной системы и ускорить загрузку в systemd-boot, вам нужно изменить параметр timeout в главном конфигурационном файле загрузчика.

20.1.2. Редактирование файла timeout

Откройте файл конфигурации загрузчика в текстовом редакторе (например, nano) с правами суперпользователя:

#bash

```
sudo nano /efi/loader/loader.conf
```

20.1.1. Если у вас раздел EFI примонтирован по стандартному пути /boot, используйте

#bash

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

20.1.2. Установка времени загрузки

Найдите строку с параметром timeout и установите желаемое время в секундах. Например, чтобы установить задержку всего в 1 секунду, укажите:

text

```
timeout 1
```

(Если вы хотите скрыть меню загрузчика и сразу загружать ОС по умолчанию, установите timeout 0).
Сохраните изменения (в nano нажмите Ctrl+O, затем Enter) и закройте файл (Ctrl+X).
Если вам понадобится зайти в меню загрузчика во время загрузки системы, просто удерживайте нажатой или периодически нажимайте клавишу **Пробел (Space)** сразу после включения компьютера.

20.2. Сокращение времени выбора grub

Чтобы сократить время ожидания меню GRUB в Arch Linux, измените значение тайм-аута в основном файле конфигурации и обновите загрузчик.

20.2.1. Редактирование файла timeout

Откройте файл конфигурации в текстовом редакторе (например, nano):

#bash

```
sudo nano /etc/default/grub
```

20.2.2. Установка времени

Найдите параметр GRUB_TIMEOUT и задайте желаемое время в секундах (например, 2):

text

```
GRUB_TIMEOUT=2
```

20.2.2.1. Сохранение изменений

Сохраните изменения и обновите файл конфигурации GRUB:

#bash

```
sudo grub-mkconfig -o /boot/grub/grub.cfg
```

20.2.3. Пропуск показа меню

Дополнительно можно включить пропуск показа меню при единственной ОС:

Раскомментируйте или добавьте строку:

text

```
GRUB_HIDDEN_TIMEOUT_QUIET=true
```

Снова выполните команду обновления:

#bash

```
sudo grub-mkconfig -o /boot/grub/grub.cfg
```

20.2.3. Пропуск задержки

Если вы хотите полностью убрать задержку, установите

text

```
GRUB_TIMEOUT=0.
```

В таком случае, чтобы открыть меню GRUB при следующей загрузке, зажмите и удерживайте клавишу **Shift (в режиме BIOS)** или **Esc (в режиме UEFI)** сразу после включения компьютера

30. Отключение службы сетевого времени

При установке без интернета в среде archiso (установочной флешке) службы синхронизации времени могут выдавать предупреждения. Чтобы полностью отключить автоматическое обновление времени и предотвратить попытки системы подключиться к сети:

30.1. Отключение NTP

Отключите службы сетевого времени, выполнив команду timedatectl:

#bash

```
timedatectl set-ntp false
```

30.2. Ручная установка

Установите нужный часовой пояс вручную. Напрямую настройте время, чтобы избежать «скачков» System time.

30.3. Отключение службы systemd-timesyncd

Если вы на этапе после pacstrap и уже вошли в систему через arch-chroot, дополнительно отключите службу systemd-timesyncd:

#bash

```
systemctl disable systemd-timesyncd
```

создание Headless Arch ISO через squashfs-tools zram-generator xorriso с отключением NTP без интернета

!!!!!!!!!! РЕДАКТИРОВАТЬ В ЭТОТ ПУНК !!!!!!!!!!!!!!!

Пункт в работе

Для создания собственного Headless Arch ISO (без графического интерфейса, для серверов или восстановления) с использованием squashfs-tools, zram-generator (для оптимизации работы в ОЗУ) и xorriso (для сборки образа), вам нужно выполнить пять основных шагов. В качестве базовой основы проще всего модифицировать официальный Arch ISO.

1. Подготовка и распаковка

Создайте рабочую директорию и структуру будущего ISO:

#bash

```
mkdir -p ~/custom_iso/{iso_root,arch_root}
```

Установите необходимые инструменты на хост-системе:

#bash

```
sudo pacman -S squashfs-tools xorriso arch-install-scripts
```

Скачайте актуальный образ с официального сайта и подготовьте директории:

#bash

```
mkdir -p /mnt/iso /mnt/new_iso
# Монтируем оригинальный ISO
mount -o loop archlinux-xxxx.xx.xx-x86_64.iso /mnt/iso
# Копируем содержимое (кроме squashfs, который находится в папке arch)
rsync -a --exclude=airootfs.sfs /mnt/iso/ /mnt/new_iso/
```

2. Настройка SquashFS образа

Распакуйте оригинальный файловый архив в отдельную временную папку, чтобы добавить

zram-generator и убрать лишнее:

#bash

```
mkdir /tmp/official_iso
sudo mount -o loop archlinux-XXXX.XX.XX-x86_64.iso /tmp/official_iso
unsquashfs -d ~/custom_iso/arch_root
/tmp/official_iso/arch/x86_64/airootfs.sfs
sudo umount /tmp/official_iso
```

Используйте код с осторожностью.Переключитесь в окружение (chroot) для установки нужных пакетов и настройки zram-generator:

#bash

```
mount -t proc /proc /mnt/squashfs/proc
mount --rbind /sys /mnt/squashfs/sys
mount --rbind /dev /mnt/squashfs/dev
chroot /mnt/squashfs
```

3. Настройка zram-generator и отключение NTP

3.1 Вход в окружение (Chroot):

#bash

```
sudo arch-chroot ~/custom_iso/arch_root
```

3.1.1. установите пакет:

#bash

```
pacman -S zram-generator
```

3.1.2. Создайте конфигурационный файл для автоматического создания сжатого раздела подкачки в оперативной памяти:

#bash

```
cat <<EOF > /etc/systemd/zram-generator.conf
```

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

3.1.3. Включите службу:

#bash

```
systemctl enable systemd-zram-setup@zram0
```

3.1.4. Настройте Headless (безголовый) доступ:

3.1.4.1. настройте сеть (например, через systemd-networkd и sshd).

3.1.4.2. Включите автозапуск SSH:

#bash

```
systemctl enable sshd.
# Разрешите вход root по SSH (опционально, для теста)
echo "PermitRootLogin yes" >> /etc/ssh/sshd_config
```

3.2. Отключение NTP:

Для полной независимости от сети отключите службу синхронизации времени.

#bash

```
systemctl disable systemd-timesyncd.service
timedatectl set-ntp false
```

3.3. Выйдите из окружения:

#bash

```
exit.
```

4. Упаковка SquashFS

Упакуйте измененную систему обратно в сжатый образ squashfs. Использование многопоточности ускорит процесс.

#bash

```
mkdir -p ~/custom_iso/iso_root/arch/x86_64/
sudo mksquashfs ~/custom_iso/arch_root
~/custom_iso/iso_root/arch/x86_64/airootfs.sfs -comp zstd -b 1M
```

5. Сборка ISO через xorriso

Скопируйте загрузочные файлы (ядро, initramfs, syslinux/grub конфигурацию) из оригинального ISO в ~/custom_iso/iso_root/. После этого соберите гибридный ISO-образ, готовый к записи на флешку:

#bash

```
xorriso -as mkisofs \
  -iso-level 3 \
  -full-iso9660-filenames \
  -volid "ARCH_CUSTOM" \
  -eltorito-boot isolinux/isolinux.bin \
  -eltorito-catalog isolinux/boot.cat \
  -no-emul-boot -boot-load-size 4 -boot-info-table \
  -isohybrid-mbr ~/custom_iso/iso_root/isolinux/isohdpfx.bin \
  -eltorito-alt-boot \
  -e efi/archiso/efiboot.img \
  -no-emul-boot -isohybrid-gpt-basdat \
  -output ~/custom_iso/archlinux-headless.iso \
  ~/custom_iso/iso_root/
```

6 Выгрузка iso в виндовс

7 Монтаж в hyper-V

10. Консервация операционной системы в SquashFS

10.1. Изоляция дисковой разметки хоста

создание бэкапа и полное обнуление файла /etc/fstab во избежание конфликта UUID при live-загрузке.

10.2. Проверка перед

10.2.1. Проветить тайм зону

Проверить таймзону (часовой пояс) в Arch Linux можно за пару секунд. Самый быстрый и современный способ — использовать утилиту timedatectl, которая сразу покажет текущее время и настройки

10.2.1.1. Быстрая проверка

Откройте терминал и выполните:

#bash

```
timedatectl
```

В выводе найдите строки:

- Time zone: — ваш текущий часовой пояс (например, Europe/Moscow).
- NTC service: — статус автоматической синхронизации времени

10.2.1.2. Быстрая проверка

#bash

```
date
```

Она выведет текущую дату, время и аббревиатуру часового пояса (например, MSK).

10.2.1.3. Изменить таймзону

Посмотрите список доступных часовых поясов:

#bash

```
timedatectl list-timezones
```

10.2.1.4. Установите нужный пояс

(замените Europe/Moscow на ваш регион)

#bash

```
sudo timedatectl set-timezone Europe/Moscow
```

- Проверить версии linux
- Драйверы сетевой карты реального сервера

Драйверы сетевой карты реального сервера

В виртуальной машине Hyper-V используется виртуальный сетевой адаптер (обычно dec21140 или синтетический от Microsoft). На реальном сервере будет стоять физический чип (Intel, Realtek или Broadcom и т.д.).

Что проверить:

Убедитесь, что в вашей системе tom_1 установлен пакет linux-firmware.

Команда для проверки

#bash

```
pacman -Q linux-firmware
```

Если его нет, обязательно установите (sudo pacman -S linux-firmware), иначе реальный сервер после загрузки с ISO просто не увидит свою физическую сетевую карту.

Данные для проверки по железу серверов (1 Supermicro, 1 старый HP и два ноунейм-самосбора разных поколений), а значит универсальность сетевого конфига становится задачей номер один. На таком железа дефолтные предсказуемые имена интерфейсов от systemd гарантированно разъедутся в разные стороны:

- На Supermicro это, скорее всего, будут имена вида eno1 (onboard) или пути вроде enp3s0f0.
- На старом HP интерфейсы могут обозваться как eno49, ens1 или упасть в классический

eth0.
- На ноунеймах (самосборах) всё зависит от логики материнской платы (чисто по PCI-пути — например, enp2s0).

Поэтому зашивать маску Name=en* eth* в конфиг systemd-networkd — это единственное спасение, чтобы один и тот же ISO-образ молча поднял сеть на любой из этих плат.

- Изменить временно на 192.168.1.150

Создаем конфигурациюсети сети будущего сервера

Чтобы не гадать, как система назовет сетевую карту на разном железе (eno1, enp3s0, eth0), мы заставим systemd-networkd применять настройки к любому проводному интерфейсу.
Выполните команду на tom_1 для создания конфигурационного файла:

#bash

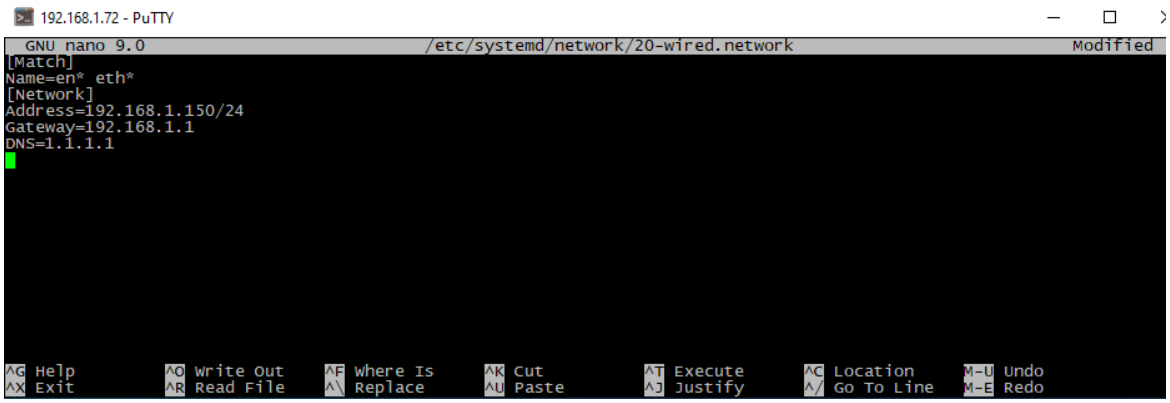
```
sudo nano /etc/systemd/network/20-wired.network
```

Вставьте в него следующие строки:

ini

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

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

Примечание: Если в вашей локальной Windows-сети используется другой поддиапазон (например, 192.168.0.X), измените IP-адрес Address и шлюз Gateway под свою рабочую сеть.



Разбор конфига на скриншоте

Всё в точности, как надо:

- Name=en* eth* — заматчит любую сетевуху.
- Address=192.168.1.150/24 — статический IP, который мы будем пинговать на новом сервере.
- Gateway=192.168.1.1 — шлюз.
- DNS=1.1.1.1 — DNS-сервер.

В правом верхнем углу горит надпись Modified. Это значит, что изменения внесены, но файл еще не сохранен на диск. *Примечание: CTRL+O для записи файла Enter для подтверждения имени файла CTRL+X для выхода из редактора nano* Как только выйдете из редактора, нам нужно убедиться, что systemd-networkd вообще включен и подхватит этот конфиг при старте образа.
Выполните команду в консоли:

#bash

```
systemctl is-enabled systemd-networkd
```

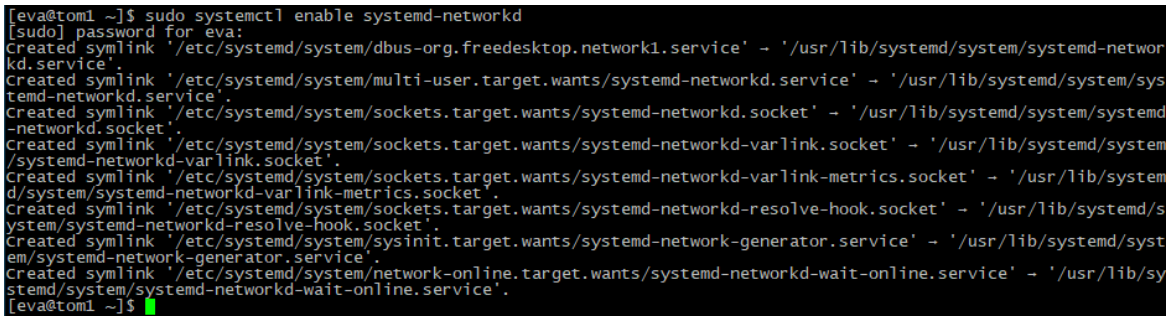
Примечание: в нашем случае служба systemd-networkd отключена (disabled)

Включаем сетевую службу

Выполните команду, чтобы активировать автозапуск сети:

#bash

```
sudo systemctl enable systemd-networkd
```



На скриншоте чётко видно, что система создала все необходимые привязки для systemd-networkd.

Теперь переходим к проверке DNS-резолвера и удаленного доступа по SSH. Без этого сервер не сможет преобразовывать имена сайтов, а мы не сможем зайти на него удаленно.

- Проверить автозыпуск ssh

Проверяем, включен ли автозапуск службы SSH:

#bash

```
systemctl is-enabled sshd
```

DNS включен, SSH в статусе enabled. Это значит, что доступ к консоли по сети у нас будет сразу после старта «вслепую».
Теперь проверяем последний критически важный сервис — веб-сервер Nginx, ради которого всё и затевается. Нам нужно убедиться, что он тоже поднимется сам.

- Проверить автозапуск nginx

Проверяем Nginx

Выполните команду в терминале:

#bash

```
systemctl is-enabled nginx
```

Проверяем, слушает ли Nginx порт 7000

Убедимся, что в конфигах нет синтаксических ошибок и порт 5000 реально прописан.

Выполните команду:

#bash

```
sudo nginx -t && grep -r "7000" /etc/nginx/
```

```
[eva@tom1 ~]$ systemctl is-enabled nginx
enabled
[eva@tom1 ~]$ sudo nginx -t && grep -r "5000" /etc/nginx/
[sudo] password for eva:
2026/05/22 20:18:13 [warn] 818#818: could not build optimal types_hash, you should increase either types_hash_max_siz
e: 1024 or types_hash_bucket_size: 64; ignoring types_hash_bucket_size
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
/etc/nginx/nginx.conf:        listen       5000;
[eva@tom1 ~]$
```

Nginx проверен, тут полный порядок! Тест конфигурации успешный (syntax is ok, test is successful), и порт 5000 чётко прописан в основном файле /etc/nginx/nginx.conf. Варнинг про types_hash — это стандартная безобидная ругань Nginx на мелкий дефолтный размер хэш-таблицы типов файлов в Arch, на работу веб-сервера он никак не повлияет.
Теперь проверяем вторую половину вашей задумки — пароли пользователей. Это критически важно, так как без паролей SSH-сервер нас просто отошьёт.

- Проверить автозапуск PHP-FPM

8.2.1.1. Проверяем статус автозапуска службы php-fpm

Убедимся, что служба PHP-FPM активирована в автозапуске, чтобы при старте флешки в ОЗУ она запустилась сама вместе с Nginx. Убедимся, что система теперь рапортует правильный статус.

#bash

```
systemctl is-enabled php-fpm
```

```
[eva@tom1 ~]$ systemctl is-enabled php-fpm
enabled
[eva@tom1 ~]$
```

(Примечание: статус enabled - включена)

- Выводим хэши паролей root и eva

Выводим хэши паролей root и eva

Выводим хэши паролей root и eva

#bash

```
sudo getent shadow root eva | cut -d: -f1,2
```

```
[eva@tom1 ~]$ sudo getent shadow root eva | cut -d: -f1,2
root:$y$j9T$.UtQy5qx7EJbR8eQoPnmR.$aAD2CoNq.XSFF1Js9h68cKmE5CRahXXDjAyK.NPnuU9
eva:$y$j9T$XQsIjp.dE297cMeuUpNkSO$g5PO2vn/O7lUzTbFb/yr9CFmPVqvP9KbaQMKRNOYlSA
[eva@tom1 ~]$
```

На скриншоте мы видим современные, надежные хэши паролей yescrypt (начинаются с y) как для root, так и для пользователя eva. Пароли заданы железно, а значит, SSH-сервер без проблем пустит вас в консоль на реальной машине. Полноценный аудит пройден, все критические точки проверены.Время переходить к финальной фазе сборки нашего автономного ISO. Погнали по шагам, аккуратно работаем с таблицей разделов.

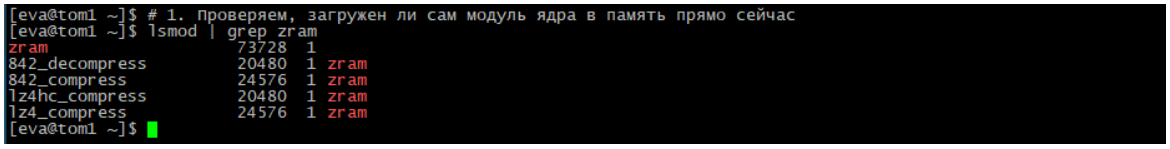
- Проверка zram-generator

Проверяем наличие ZRAM

ZRAM в Arch Linux (и других дистрибутивах) — это модуль ядра Linux, который создает виртуальный диск в оперативной памяти и сжимает данные на лету. Давайте посмотрим, активен ли этот модуль ядра на вашей чистой, только что обновленной системе tom_1. Выполните в консоли tom_1 две диагностические команды:

#bash

```
# 1. Проверяем, загружен ли сам модуль ядра в память прямо сейчас
lsmod | grep zram
```

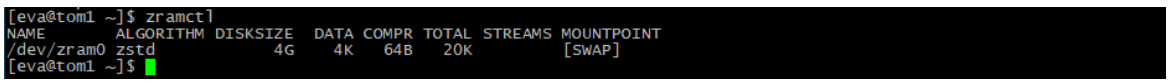


Проверка параметров активных дисков ZRAM

Выполните в терминале tom_1 ровно одну команду, чтобы увидеть физический размер и алгоритм сжатия созданного диска подкачки:

#bash

```
# 1. Проверяем, загружен ли сам модуль ядра в память прямо сейчас
zramctl
```



Установка генератора ZRAM

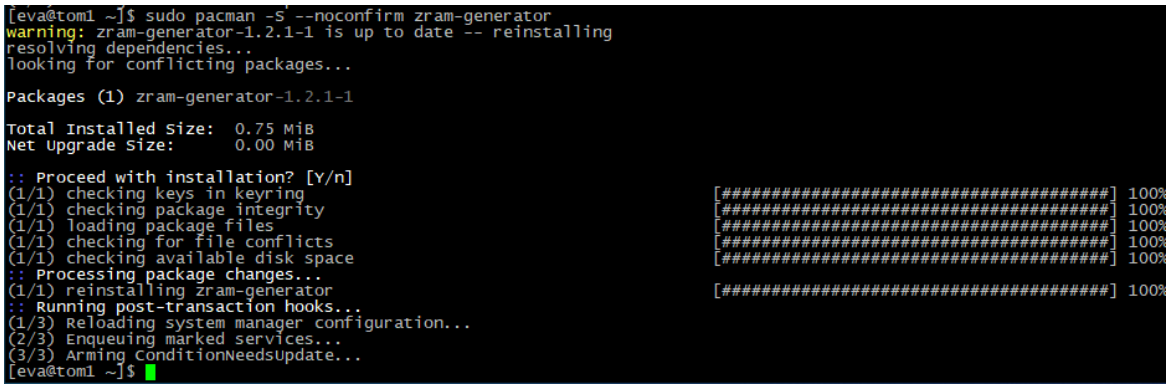
Ставим инструменты zram-generator

zram-generator в Arch Linux — это утилита, которая автоматически создает и настраивает диски zram (сжатая оперативная память) для использования в качестве очень быстрого раздела подкачки (swap).
Установим пакет squashfs-tools, в который и входит нужная команда.
Выполняйте:

#bash

```
sudo pacman -S --noconfirm zram-generator
```

Флаги -S (Синхронизация / Установка) и –noconfirm (Без подтверждения)



Проверка установки пакетов

#bash

```
pacman -Q zram-generator
```

(*Если пакеты установлены, терминал выведет их версии, иначе, вы получите ошибку (например, error: package 'squashfs-tools' was not found).*)

- бекап sfid

Временное обнуление fstab

О файле

Файл /etc/**fstab** (от File Systems Table) — это конфигурационный файл, который хранит информацию о разделах диска, флешках и сетевых хранилищах, и указывает системе, как именно и куда их нужно монтировать при запуске.

Предназначение

- Автозагрузка: Без него операционная система не поймет, где искать корневой раздел / и другие важные системные диски, и просто не сможет загрузиться.
- Параметры монтирования: Он определяет права доступа (например, можно ли запускать исполняемые файлы с конкретного диска) и режим работы (чтение или чтение и запись).
- Организация: Позволяет привязать жесткие диски, SSD или разделы для файлов подкачки (swap) к нужным папкам.

В других дистрибутивах Linux этот файл создается автоматически при установке. В Arch Linux процесс установки выполняется вручную, поэтому там его чаще всего генерируют с помощью специальной команды: genfstab -U /mnt » /mnt/etc/fstab.

Структура файла

Файл состоит из строк, разделенных пробелами или табуляцией. Каждая строка описывает одно устройство и состоит из 6 колонок:

1. Устройство (Block device): Обычно здесь указывается уникальный идентификатор диска (UUID), чтобы система не запуталась, если вы поменяете порты подключения (например, UUID=1234-abcd).
2. Точка монтирования (Mount point): Папка в системе, куда «подключается» диск (например, /, /home или /mnt/games).
3. Файловая система (FSType): Тип файловой системы (например, ext4, btrfs, xfs или vfat).
4. Параметры (Mount options): Инструкции по работе с диском (например, defaults, noatime, ro — только для чтения).
5. Резервная копия (Dump): Флаг для утилиты резервного копирования (обычно 0).
6. Проверка диска (Pass): Очередность проверки диска утилитой fsck при загрузке (корень — 1, остальные диски — 2, отключено — 0).

Подробное руководство по редактированию и настройке параметров можно изучить на официальной ArchWiki: fstab.

Безопасность операции с fstab

Сначала сделаем бэкап, проверим его и fstab на tom_1, чтобы живой образ загружался целиком в оперативную память и не пытался монтировать несуществующие диски.
Выполните по очереди эти три команды:

#bash

```
sudo cp /etc/fstab /etc/fstab.bak
```

Проверяем наличие резервной копии fstab

#bash

```
ls -la /etc/fstab /etc/fstab.bak
```

- Стереть sfid

Очищаем оригинальный файл fstab

Внимание: после использования утилиты сжатая файловой система SquashFS и копирования слепка, не забываем вернуть конфигурационный файл fstab из бекапа

#bash

```
sudo truncate -s 0 /etc/fstab
```

Контрольная проверка очищения fstab

Фиксируем, что вывод чтения оригинального файла пуст

#bash

```
cat /etc/fstab
```

Контроль: Команда cat должна вернуть абсолютно пустую строку.

10.3. Генерация монолитного SquashFS-слепка

запаковка корневой системы утилитой mksquashfs с исключениями виртуальных директорий.

Создание airootfs.sfs

Утилита mksquashfs

mksquashfs — это консольная утилита в Arch Linux, предназначенная для создания сжатых файловых систем SquashFS, которые работают только для чтения. Она является частью пакета squashfs-tools и чаще всего используется для создания Live USB, архивации системы и упаковки портативного софта

Применение

- Резервные копии: Позволяет упаковать всю систему или отдельные директории в один компактный образ без изменения прав доступа и символических ссылок.
- Создание Live-образов: С помощью mksquashfs создаются загрузочные модули (например, для archiso или сборок на его базе), которые можно распаковывать и запускать в ОЗУ.
- Портативные приложения (AppImage): Формат AppImage по своей сути является упакованным образом SquashFS

Преимущества

- Высокая степень сжатия: Поддерживает современные алгоритмы, включая xz, gzip, lz4 и zstd (по умолчанию).
- Прямой доступ: ОС может монтировать этот файл как диск и читать данные на лету без предварительной полной распаковки всего архива.

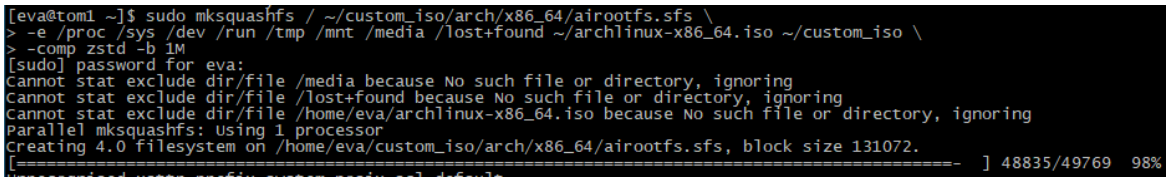
Запаковываем систему в SquashFS

Теперь запускаем самую ресурсоемкую команду, которая заморозит систему со всеми нашими универсальными сетевыми конфигами, паролями и Nginx. Она проигнорирует саму себя, бэкапы и виртуальный мусор.
Скопируйте и вставьте в терминал:

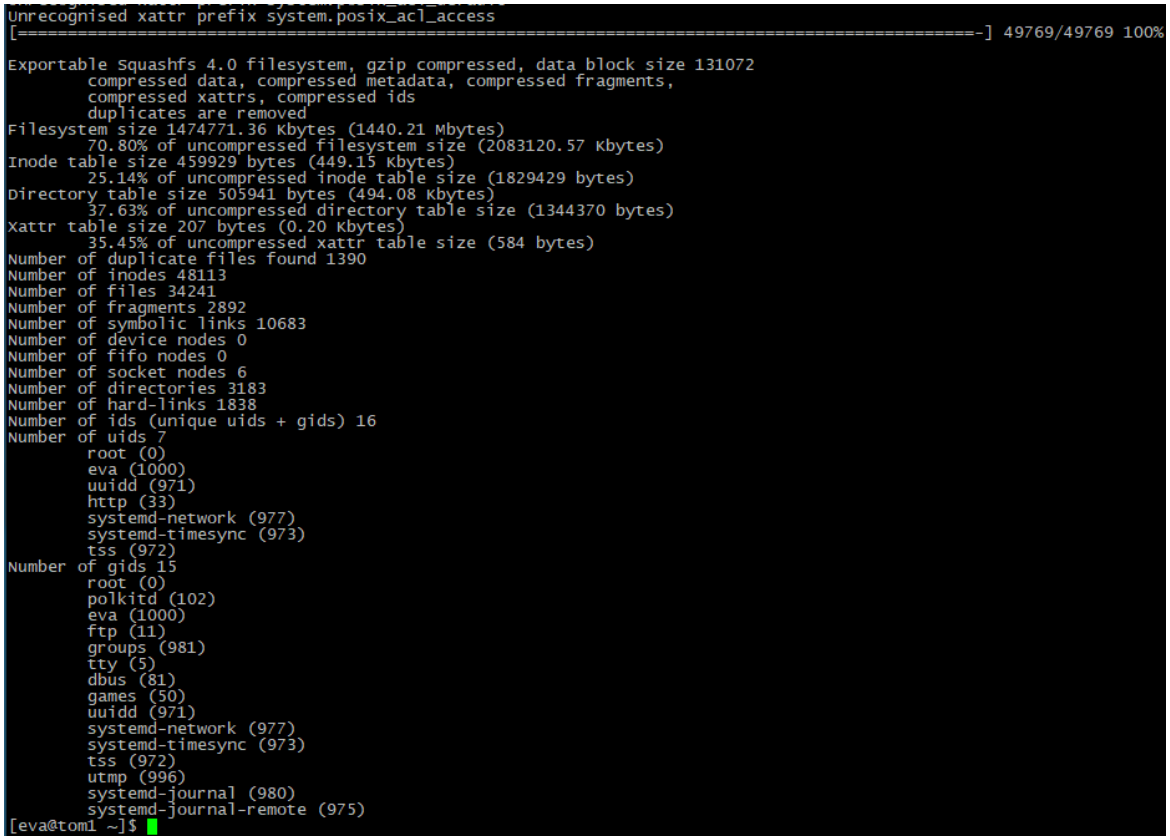
#bash

```
sudo mksquashfs / ~/custom_iso/arch/x86_64/airootfs.sfs \
-e /proc /sys /dev /run /tmp /mnt /media /lost+found ~/archlinux-
x86_64.iso ~/custom_iso \
```

```
-comp zstd -b 1M
```



(Процесс займет несколько минут. Пока он идёт, терминал будет занят.



Ждём полного завершения, пока не появится строка [eva@tom1 ~]$).

10.4. Немедленное восстановление хоста

Немедленно возвращаем fstab на место

Как только команда mksquashfs полностью отработает и вернет вам управление, немедленно и без пауз выполните команду восстановления оригинальной таблицы разделов:

#bash

```
sudo mv /etc/fstab.bak /etc/fstab
```

```
[eva@tom1 ~]$ sudo mv /etc/fstab.bak /etc/fstab
[sudo] password for eva:
[eva@tom1 ~]$
```

Контрольная проверка fstab

Убедимся, что файл вернулся и внутри него снова прописаны диски текущей виртуалки:

#bash

```
cat /etc/fstab
```

(Вы должны увидеть строчки с монтированием ваших UUID или разделов для /, /boot и т.д. Если текст появился — tom_1 в полной безопасности, можно выдохнуть).

```
[eva@tom1 ~]$ cat /etc/fstab
# Static information about the filesystems.
# See fstab(5) for details.

# <file system> <dir> <type> <options> <dump> <pass>
# /dev/sdb2
UUID=53b73831-4cdb-4783-8dd1-e2342ec6c2bf       /               btrfs           rw,relatime,discard=async,space_cache
=v2,subvol=/@   0 0

# /dev/sdb2
UUID=53b73831-4cdb-4783-8dd1-e2342ec6c2bf       /home           btrfs           rw,relatime,discard=async,space_cache
=v2,subvol=/@home       0 0

# /dev/sdb2
UUID=53b73831-4cdb-4783-8dd1-e2342ec6c2bf       /var/cache/pacman/pkg   btrfs           rw,relatime,discard=async,spa
ce_cache=v2,subvol=/@pkg        0 0

# /dev/sdb2
UUID=53b73831-4cdb-4783-8dd1-e2342ec6c2bf       /var/log        btrfs           rw,relatime,discard=async,space_cache
=v2,subvol=/@log        0 0

# /dev/sdb1
UUID=64D4-DC3F          /boot           vfat            rw,relatime,fmask=0022,dmask=0022,codepage=437,iocharset=asci
i,shortname=mixed,utf8,errors=remount-ro        0 2

[eva@tom1 ~]$
```

На скриншоте чётко видно, что все btrfs-субтома (/@, /@home, /@pkg, /@log) и UEFI-раздел /boot вернулись на свои места. Теперь система гарантированно перезагрузится без сбоев.
Смена владельца файла слепка мы тоже сделали (sudo chown eva:eva ...).

===== Изменить постоянно на 192.168.1.72=====

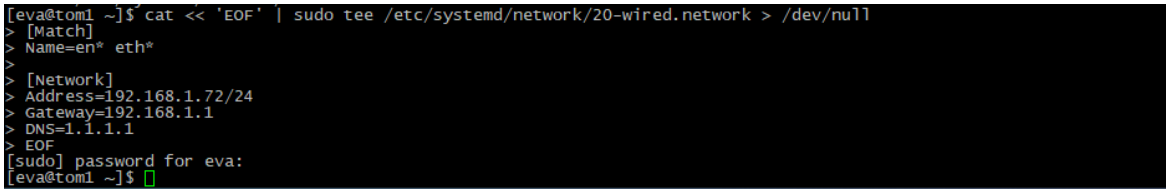
7.3.2. Перезапись файла 20-wired.network на IP 192.168.1.72

Мы используем монолитную команду cat « 'EOF', которая полностью затрёт старый конфиг и запишет новый чистый текст. Это исключает ошибки ручного ввода в редакторах.
Выполните в терминале команду:

#bash

```
cat << 'EOF' | sudo tee /etc/systemd/network/20-wired.network >
/dev/null
[Match]
Name=en* eth*
```

```
[Network]
Address=192.168.1.72/24
Gateway=192.168.1.1
DNS=1.1.1.1
EOF
```

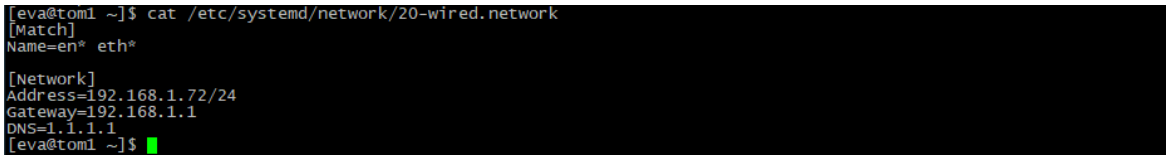


Перед тем как перезапустить сеть, мы обязаны сделать шаг контроля и убедиться, что файл перезаписался именно так, как нам нужно.

7.3.3. Проверка измененного файла 20-wired.network

#bash

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



(Внутри прописан наш целевой статический адрес 192.168.1.72.)

7.3.4. Применение настроек сети

Чтобы система сбросила старый адрес 192.168.1.150 и применила новый, нам необходимо полностью перезапустить сетевую службу systemd-networkd.

#bash

```
sudo systemctl restart systemd-networkd
```

Важно: Как только вы выполните эту команду, текущая SSH-сессия PuTTY сразу же прервётся (окно зависнет), так как IP-адрес машины мгновенно изменится на 192.168.1.72.

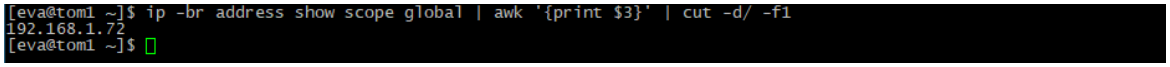
7.3.5. Проверка нового адреса

Откройте новое окно PuTTY и подключитесь к tom_1 по его новому постоянному адресу: 192.168.1.72

#bash

```
ip -br address show scope global | awk '{print $3}' | cut -d/ -f1
```

(Система отобразила наш новый постоянный IP-адрес 192.168.1.72)



11. Формирование структуры ISO и UEFI-загрузчика

11.1. Создание дерева каталогов загрузчика

развертывание путей EFI внутри директории конструктора.

11.2. Импорт оригинальных файлов загрузки

перенос бинарника BOOTX64.EFI, ядра и initramfs из хост-системы.

11.3. Конфигурация загрузчика systemd-boot

привязка загрузки к глобальной метке archisolabel=ARCH_202605, активация вывода в COM-порт и отключение прерываний Hyper-V.

12. Финальная сборка, перенос и тестирование образа

12.1. Компиляция универсального ISO

упаковка папки конструктора через xorriso с флагами гибридной разметки и жестким указанием -volid.

12.2. Экспорт ISO-образа в Windows

скачивание готового файла на рабочую станцию средствами PowerShell-команды scp.

12.3. Запись на физический носитель и развертывание в Hyper-V

инструкция по прожигу флешки в Rufus (GPT/UEFI) и запуску на изолированной виртуальной машине tom_2.

!!!!!!!!!!! переходим к iso !!!!!!!!!!!

!!!!! Убрать перед сборкой iso !!!!!!!

Драйверы сетевой карты реального сервера

В виртуальной машине Hyper-V используется виртуальный сетевой адаптер (обычно dec21140 или синтетический от Microsoft). На реальном сервере будет стоять физический чип (Intel, Realtek или Broadcom и т.д.).

Что проверить:

Убедитесь, что в вашей системе tom_1 установлен пакет linux-firmware.

Команда для проверки

#bash

```
pacman -Q linux-firmware
```

Если его нет, обязательно установите (sudo pacman -S linux-firmware), иначе реальный сервер после загрузки с ISO просто не увидит свою физическую сетевую карту.

Данные для проверки по железу серверов (1 Supermicro, 1 старый HP и два ноунейм-самосбора разных поколений), а значит универсальность сетевого конфига становится задачей номер один. На таком железа дефолтные предсказуемые имена интерфейсов от systemd гарантированно разъедутся в разные стороны:

- На Supermicro это, скорее всего, будут имена вида eno1 (onboard) или пути вроде enp3s0f0.
- На старом HP интерфейсы могут обозваться как eno49, ens1 или упасть в классический eth0.

- На ноунеймах (самосборах) всё зависит от логики материнской платы (чисто по PCI-пути — например, enp2s0).

Поэтому зашивать маску Name=en* eth* в конфиг systemd-networkd — это единственное спасение, чтобы один и тот же ISO-образ молча поднял сеть на любой из этих плат.

Создаем конфигурациюсети сети будущего сервера

Чтобы не гадать, как система назовет сетевую карту на разном железе (eno1, enp3s0, eth0), мы заставим systemd-networkd применять настройки к любому проводному интерфейсу.
Выполните команду на tom_1 для создания конфигурационного файла:

#bash

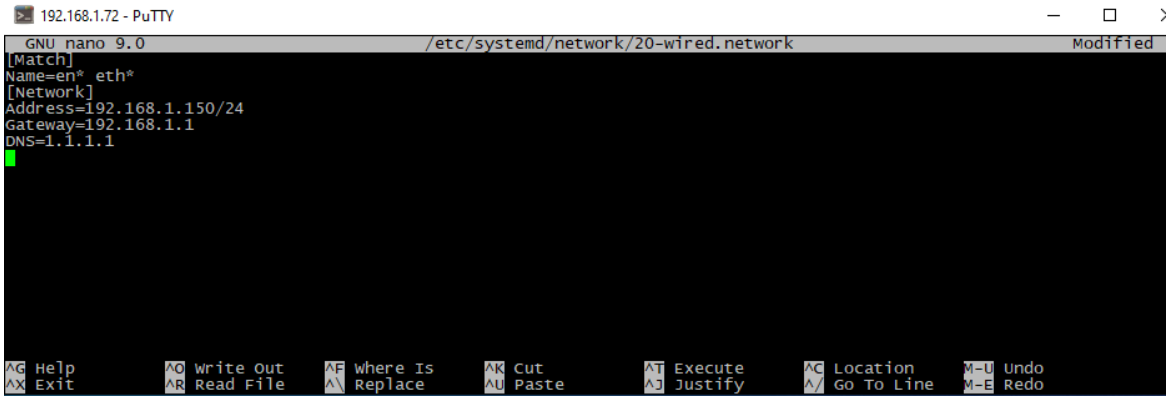
```
sudo nano /etc/systemd/network/20-wired.network
```

Вставьте в него следующие строки:

ini

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

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

Примечание: Если в вашей локальной Windows-сети используется другой поддиапазон (например, 192.168.0.X), измените IP-адрес Address и шлюз Gateway под свою рабочую сеть.



Разбор конфига на скриншоте

Всё в точности, как надо:

- Name=en* eth* — заматчит любую сетевуху.
- Address=192.168.1.150/24 — статический IP, который мы будем пинговать на новом сервере.
- Gateway=192.168.1.1 — шлюз.
- DNS=1.1.1.1 — DNS-сервер.

В правом верхнем углу горит надпись Modified. Это значит, что изменения внесены, но файл еще не сохранен на диск. *Примечание: CTRL+O для записи файла Enter для подтверждения имени файла CTRL+X для выхода из редактора nano* Как только выйдете из редактора, нам нужно убедиться, что systemd-networkd вообще включен и подхватит этот конфиг при старте образа.
Выполните команду в консоли:

#bash

```
systemctl is-enabled systemd-networkd
```

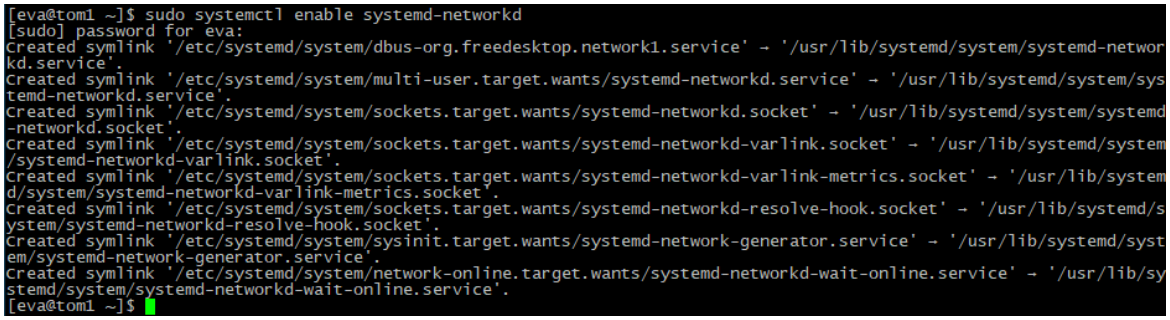
Примечание: в нашем случае служба systemd-networkd отключена (disabled)

Включаем сетевую службу

Выполните команду, чтобы активировать автозапуск сети:

#bash

```
sudo systemctl enable systemd-networkd
```



На скриншоте чётко видно, что система создала все необходимые привязки для systemd-networkd.
Теперь переходим к проверке DNS-резолвера и удаленного доступа по SSH. Без этого сервер не сможет преобразовывать имена сайтов, а мы не сможем зайти на него удаленно.

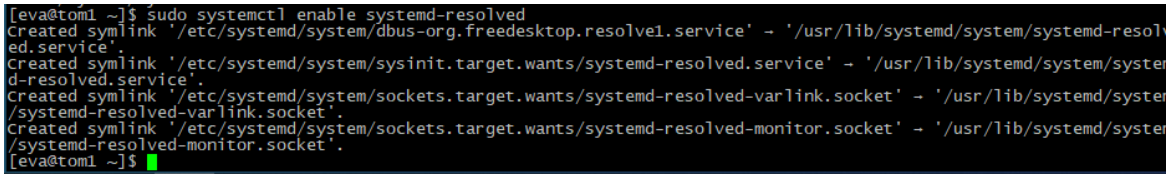
Включаем DNS и проверяем SSH

Выполните в терминале по очереди следующие две команды:

Включаем службу резолвинга имён (DNS):

#bash

```
sudo systemctl enable systemd-resolved
```



Проверяем, включен ли автозапуск службы SSH:

#bash

```
systemctl is-enabled sshd
```

DNS включен, SSH в статусе enabled. Это значит, что доступ к консоли по сети у нас будет сразу после старта «вслепую».
Теперь проверяем последний критически важный сервис — веб-сервер Nginx, ради которого всё и затевается. Нам нужно убедиться, что он тоже поднимется сам.

Проверяем Nginx

Выполните команду в терминале:

#bash

```
systemctl is-enabled nginx
```

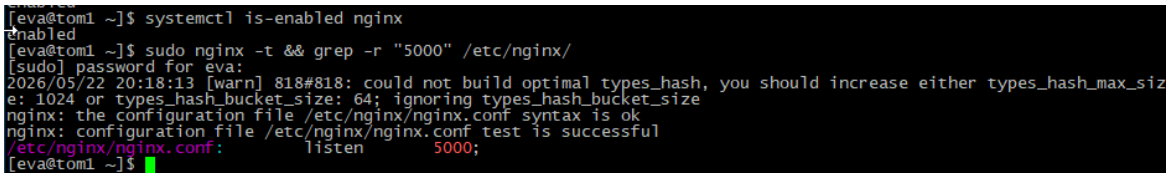
Проверяем, слушает ли Nginx порт 7000

Убедимся, что в конфигах нет синтаксических ошибок и порт 5000 реально прописан.

Выполните команду:

#bash

```
sudo nginx -t && grep -r "7000" /etc/nginx/
```



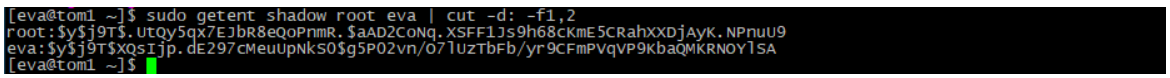
Nginx проверен, тут полный порядок! Тест конфигурации успешный (syntax is ok, test is successful), и порт 5000 чётко прописан в основном файле /etc/nginx/nginx.conf. Варнинг про types_hash — это стандартная безобидная ругань Nginx на мелкий дефолтный размер хэш-таблицы типов файлов в Arch, на работу веб-сервера он никак не повлияет.
Теперь проверяем вторую половину вашей задумки — пароли пользователей. Это критически важно, так как без паролей SSH-сервер нас просто отошьёт.

Выводим хэши паролей root и eva

Выводим хэши паролей root и eva

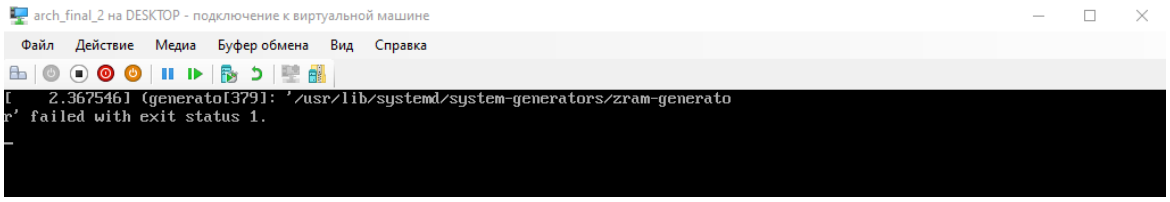
#bash

```
sudo getent shadow root eva | cut -d: -f1,2
```



На скриншоте мы видим современные, надежные хэши паролей yescrypt (начинаются с y) как для root, так и для пользователя eva. Пароли заданы железно, а значит, SSH-сервер без проблем пустит вас в консоль на реальной машине. Полноценный аудит пройден, все критические точки проверены.Время переходить к финальной фазе сборки нашего автономного ISO. Погнали по шагам, аккуратно работаем с таблицей разделов.

Ошибка в загрузку флешки

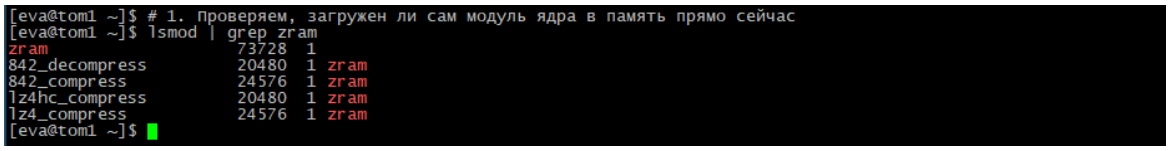


Проверяем наличие ZRAM

ZRAM в Arch Linux (и других дистрибутивах) — это модуль ядра Linux, который создает виртуальный диск в оперативной памяти и сжимает данные на лету. Давайте посмотрим, активен ли этот модуль ядра на вашей чистой, только что обновленной системе tom_1. Выполните в консоли tom_1 две диагностические команды:

#bash

```
# 1. Проверяем, загружен ли сам модуль ядра в память прямо сейчас
lsmod | grep zram
```

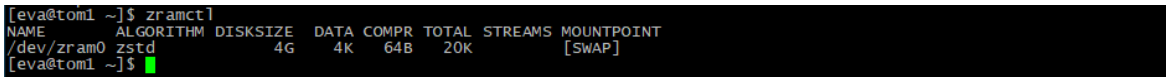


Проверка параметров активных дисков ZRAM

Выполните в терминале tom_1 ровно одну команду, чтобы увидеть физический размер и алгоритм сжатия созданного диска подкачки:

#bash

```
# 1. Проверяем, загружен ли сам модуль ядра в память прямо сейчас
zramctl
```



Установка инструментов сжатия и генератора ZRAM

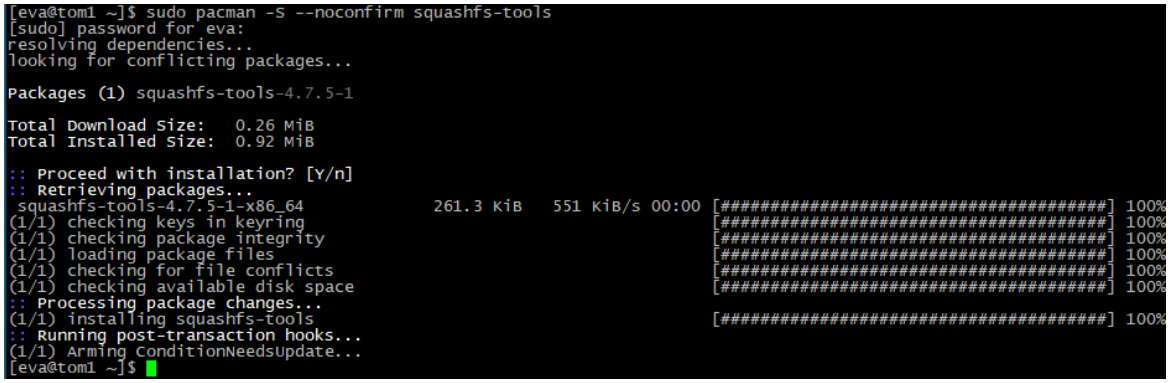
Ставим инструменты SquashFS

Пакет **squashfs-tools** в Arch Linux — это набор консольных утилит для создания, распаковки и модификации сжатых файловых систем SquashFS.
Установим пакет squashfs-tools, в который и входит нужная команда.
Выполняйте:

#bash

```
sudo pacman -S --noconfirm squashfs-tools
```

Флаги -S (Синхронизация / Установка) и –noconfirm (Без подтверждения)



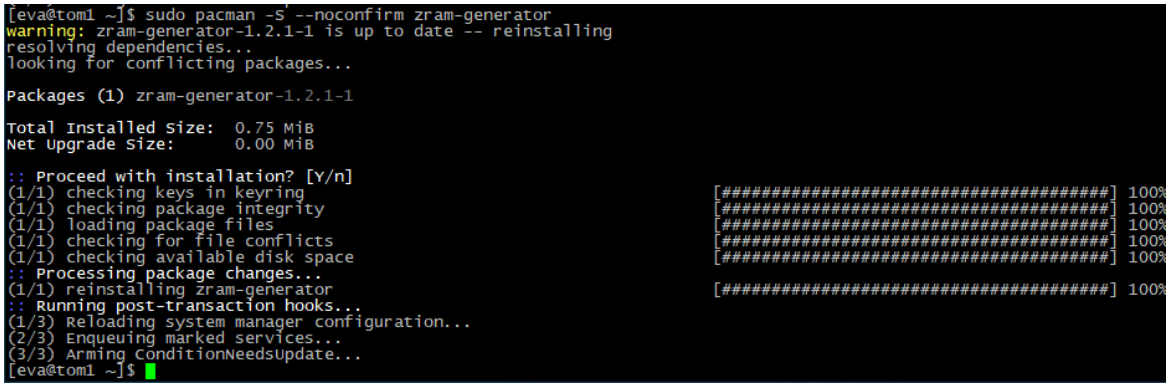
Ставим инструменты zram-generator

zram-generator в Arch Linux — это утилита, которая автоматически создает и настраивает диски zram (сжатая оперативная память) для использования в качестве очень быстрого раздела подкачки (swap).
Установим пакет squashfs-tools, в который и входит нужная команда.
Выполняйте:

#bash

```
sudo pacman -S --noconfirm zram-generator
```

Флаги -S (Синхронизация / Установка) и –noconfirm (Без подтверждения)



Проверка установки пакетов

#bash

```
pacman -Q squashfs-tools zram-generator
```

(*Если пакеты установлены, терминал выведет их версии, иначе, вы получите ошибку (например, error: package 'squashfs-tools' was not found).*)

Создание структуры каталогов для конструктора ISO

Перед тем как запустить утилиту сжатия, нам необходимо подготовить чистое рабочее дерево папок в домашней директории пользователя eva, куда мы позже разложим ядро и файлы загрузчика
Выполните в терминале:

#bash

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

(Флаг -p заставит систему создать всю цепочку папок за один раз).

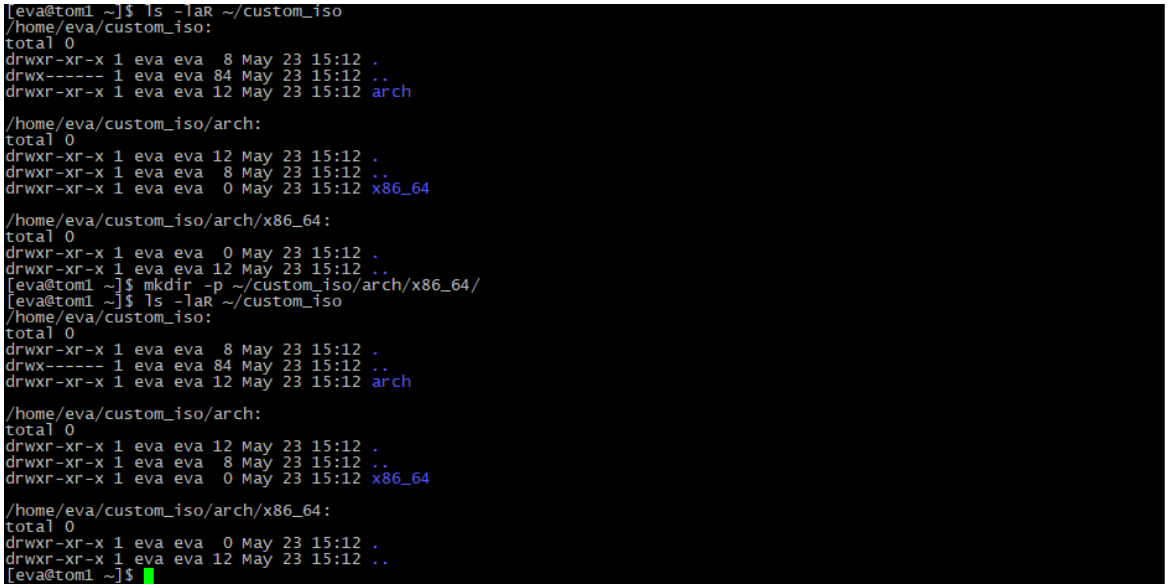
Проверка созданной структуры

Выполните в терминале команду, которая покажет полный путь и содержимое созданной

папки:

#bash

```
ls -laR ~/custom_iso
```



* //Система должна показать, что внутри `custom_iso` есть папка `arch,` а внутри неё — папка `x86_64.//`
* //Все они должны быть пустыми (внутри только точки . и ..), готовыми принять наш живой корень.//

!!!!!!!!!!!МЫ ТУТ!!!!!!!!!!!!!!!!!

Следующим пуктом Сменить статический IP адрес на 192.168.1.150 на время записи слепка, а после назад!!!

Временное обнуление fstab

О файле

Файл /etc/**fstab** (от File Systems Table) — это конфигурационный файл, который хранит информацию о разделах диска, флешках и сетевых хранилищах, и указывает системе, как именно и куда их нужно монтировать при запуске.

Предназначение

- Автозагрузка: Без него операционная система не поймет, где искать корневой раздел / и

другие важные системные диски, и просто не сможет загрузиться.
- Параметры монтирования: Он определяет права доступа (например, можно ли запускать исполняемые файлы с конкретного диска) и режим работы (чтение или чтение и запись).
- Организация: Позволяет привязать жесткие диски, SSD или разделы для файлов подкачки (swap) к нужным папкам.

В других дистрибутивах Linux этот файл создается автоматически при установке. В Arch Linux процесс установки выполняется вручную, поэтому там его чаще всего генерируют с помощью специальной команды: genfstab -U /mnt » /mnt/etc/fstab.

Структура файла

Файл состоит из строк, разделенных пробелами или табуляцией. Каждая строка описывает одно устройство и состоит из 6 колонок:

1. Устройство (Block device): Обычно здесь указывается уникальный идентификатор диска (UUID), чтобы система не запуталась, если вы поменяете порты подключения (например, UUID=1234-abcd).
2. Точка монтирования (Mount point): Папка в системе, куда «подключается» диск (например, /, /home или /mnt/games).
3. Файловая система (FSType): Тип файловой системы (например, ext4, btrfs, xfs или vfat).
4. Параметры (Mount options): Инструкции по работе с диском (например, defaults, noatime, ro — только для чтения).
5. Резервная копия (Dump): Флаг для утилиты резервного копирования (обычно 0).
6. Проверка диска (Pass): Очередность проверки диска утилитой fsck при загрузке (корень — 1, остальные диски — 2, отключено — 0).

Подробное руководство по редактированию и настройке параметров можно изучить на официальной ArchWiki: fstab.

Безопасность операции с fstab

Сначала сделаем бэкап, проверим его и fstab на tom_1, чтобы живой образ загружался целиком в оперативную память и не пытался монтировать несуществующие диски.
Выполните по очереди эти три команды:

#bash

```
sudo cp /etc/fstab /etc/fstab.bak
```

Проверяем наличие резервной копии fstab

#bash

```
ls -la /etc/fstab /etc/fstab.bak
```

Очищаем оригинальный файл fstab

Внимание: после использования утилиты сжатая файловой система SquashFS и копирования слепка, не забываем вернуть конфигурационный файл fstab из бекапа

#bash

```
sudo truncate -s 0 /etc/fstab
```

Контрольная проверка очищения fstab

Фиксируем, что вывод чтения оригинального файла пуст

#bash

```
cat /etc/fstab
```

Контроль: Команда cat должна вернуть абсолютно пустую строку.

Создание airootfs.sfs

Утилита mksquashfs

mksquashfs — это консольная утилита в Arch Linux, предназначенная для создания сжатых файловых систем SquashFS, которые работают только для чтения. Она является частью пакета squashfs-tools и чаще всего используется для создания Live USB, архивации системы и упаковки портативного софта

Применение

- Резервные копии: Позволяет упаковать всю систему или отдельные директории в один компактный образ без изменения прав доступа и символических ссылок.
- Создание Live-образов: С помощью mksquashfs создаются загрузочные модули (например, для archiso или сборок на его базе), которые можно распаковывать и запускать в ОЗУ.
- Портативные приложения (AppImage): Формат AppImage по своей сути является упакованным образом SquashFS

Преимущества

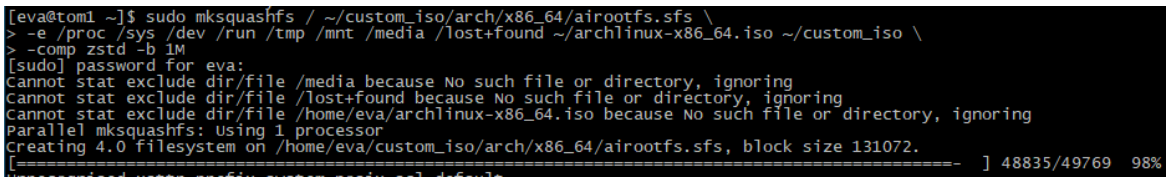
- Высокая степень сжатия: Поддерживает современные алгоритмы, включая xz, gzip, lz4 и zstd (по умолчанию).
- Прямой доступ: ОС может монтировать этот файл как диск и читать данные на лету без предварительной полной распаковки всего архива.

Запаковываем систему в SquashFS

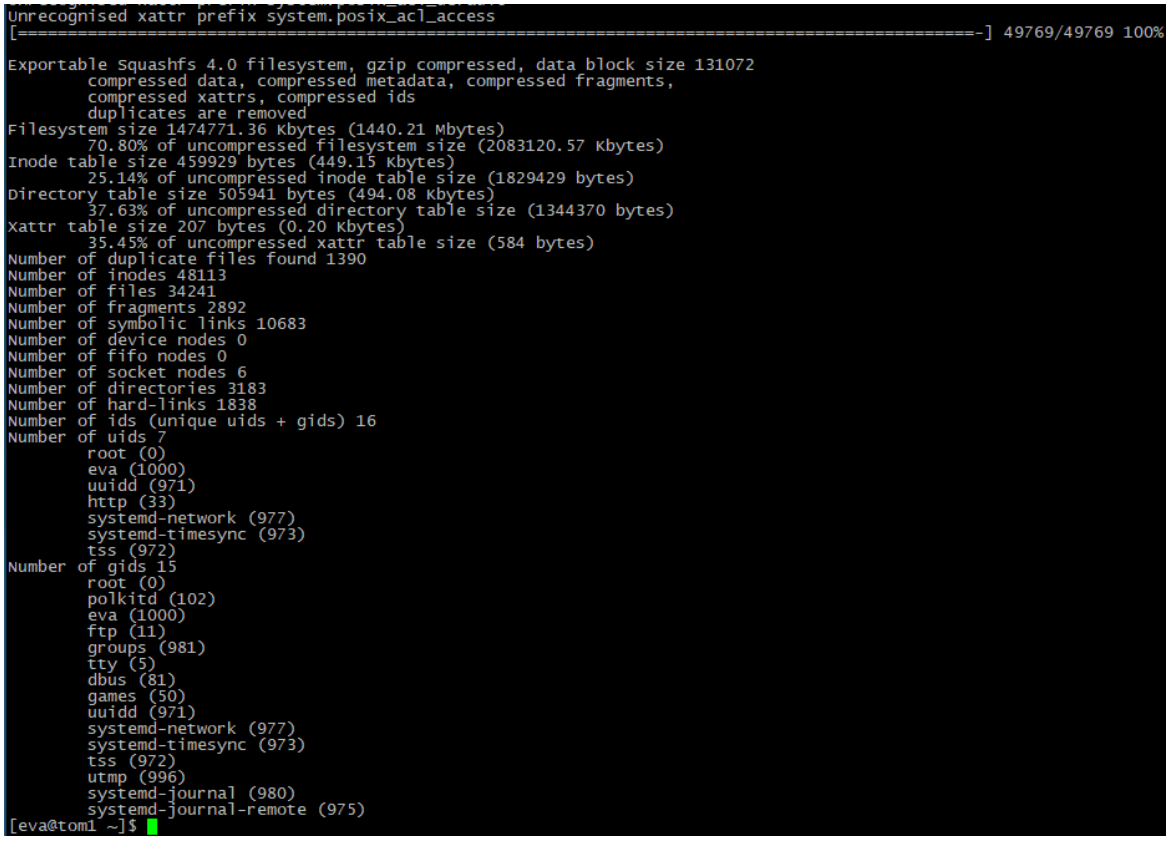
Теперь запускаем самую ресурсоемкую команду, которая заморозит систему со всеми нашими универсальными сетевыми конфигами, паролями и Nginx. Она проигнорирует саму себя, бэкапы и виртуальный мусор.
Скопируйте и вставьте в терминал:

#bash

```
sudo mksquashfs / ~/custom_iso/arch/x86_64/airootfs.sfs \
-e /proc /sys /dev /run /tmp /mnt /media /lost+found ~/archlinux-x86_64.iso ~/custom_iso \
-comp zstd -b 1M
```



(Процесс займет несколько минут. Пока он идёт, терминал будет занят.



Ждём полного завершения, пока не появится строка [eva@tom1 ~]$).

Немедленно возвращаем fstab на место

Как только команда mksquashfs полностью отработает и вернет вам управление, немедленно и без пауз выполните команду восстановления оригинальной таблицы разделов:

#bash

```
sudo mv /etc/fstab.bak /etc/fstab
```

Контрольная проверка fstab

Убедимся, что файл вернулся и внутри него снова прописаны диски текущей виртуалки:

#bash

```
cat /etc/fstab
```

(Вы должны увидеть строчки с монтированием ваших UUID или разделов для /, /boot и т.д. Если текст появился — tom_1 в полной безопасности, можно выдохнуть).

На скриншоте чётко видно, что все btrfs-субтома (/@, /@home, /@pkg, /@log) и UEFI-раздел /boot вернулись на свои места. Теперь система гарантированно перезагрузится без сбоев.
Смена владельца файла слепка мы тоже сделали (sudo chown eva:eva ...).

Следующая проблема: Скелет загрузчика UEFI

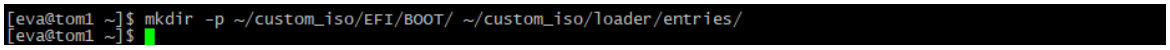
Переходим к самой важной части «слепого» взлёта — подготовке загрузчика. Чтобы материнские платы Supermicro, HP и ноунеймы поняли, как запускать наш кастомный образ, внутри папки ~/custom_iso/ должна лежать строгая структура файлов UEFI-загрузчика systemd-boot.
Сейчас у нас в ~/custom_iso/ есть только папка arch/x86_64/airootfs.sfs. Если запустить сборку ISO прямо сейчас, образ получится пустым и не загрузится ни на одном сервере.
Нам нужно скопировать бинарники загрузчика и конфиги из живой системы tom_1 прямо в наш конструктор.

Создаем структуру папок для UEFI и конфигурации

Выполните команду одной строкой:

#bash

```
mkdir -p ~/custom_iso/EFI/BOOT/ ~/custom_iso/loader/entries/
```

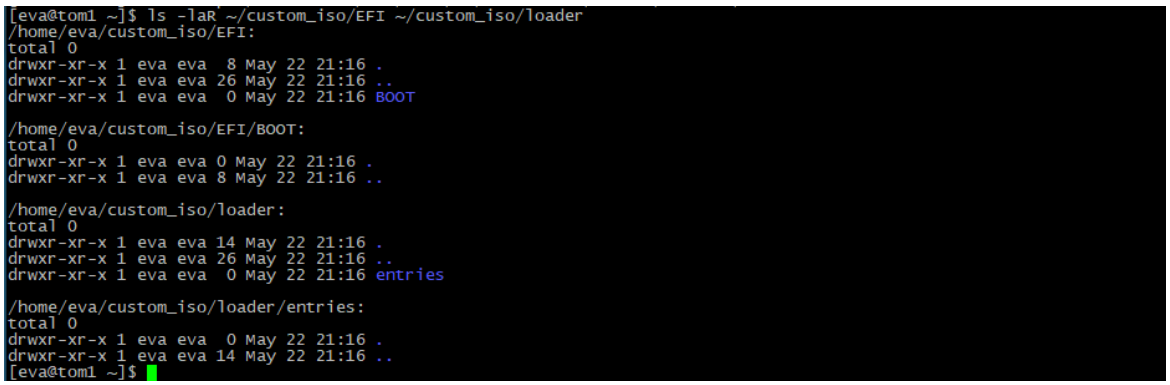


Проверка структуры UEFI

Выполните в терминале команду:

#bash

```
ls -laR ~/custom_iso/EFI ~/custom_iso/loader
```



На что смотрим в выводе:

- Система должна показать пустую папку ~/custom_iso/EFI/BOOT
- И пустую папку ~/custom_iso/loader/entries

Структура каталогов EFI/BOOT и loader/entries создана без единой ошибки, права на месте.
Скелет готов принимать файлы.
Теперь переходим к заполнению этих папок «жизненно важными органами» системы: загрузчиком UEFI, ядром Linux и виртуальным диском.

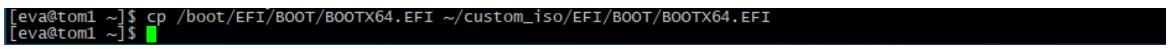
Копируем файлы загрузки

Выполните в терминале строго по очереди следующие три команды:

Копируем сам загрузчик UEFI:

#bash

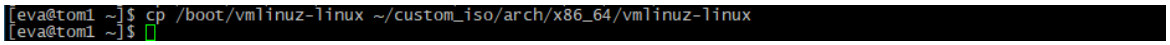
```
cp /boot/EFI/BOOT/BOOTX64.EFI ~/custom_iso/EFI/BOOT/BOOTX64.EFI
```



Копируем ядро Linux:

#bash

```
cp /boot/vmlinuz-linux ~/custom_iso/arch/x86_64/vmlinuz-linux
```

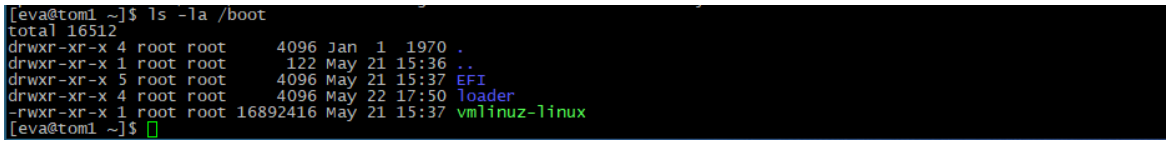


Проверяем содержимое каталога /boot

Выполните команду:

#bash

```
ls -la /boot
```



Что-то проебали и Остановились тут

Скачиваем чистый ISO

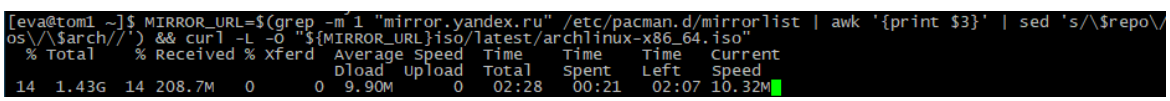
Поскольку сеть на tom_1 у нас работает, первым делом вытягиваем актуальную ссылку на зеркало Яндекса и скачиваем официальный образ Arch Linux. Из него мы и заберём правильные файлы загрузчика

Скачиваем чистый ISO Arch Linux

Выполните эти две команды, чтобы вытащить актуальную ссылку на зеркало Яндекса и скачать чистый Arch Linux:

#bash

```
MIRROR_URL=$(grep -m 1 "mirror.yandex.ru" /etc/pacman.d/mirrorlist |
awk '{print $3}' | sed 's/\$repo\/os\/\$arch//')
curl -L -O "${MIRROR_URL}iso/latest/archlinux-x86_64.iso"
```



(В консоли побегут проценты скачивания файла размером около 1.1–1.5 ГБ).

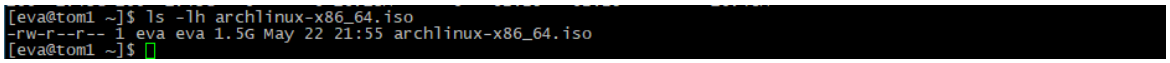
Контрольная проверка скачанного файла

Как только закачка полностью завершится, мы обязаны убедиться, что файл лёг в корень домашней директории и не является «нулевым»

Выполните команду проверки

#bash

```
ls -lh archlinux-x86_64.iso
```



Нам нужно смонтировать этот ISO во временную директорию и вытащить оттуда оригинальные папки EFI и loader. Папку arch копировать НЕ будем, чтобы случайно не затереть наш кастомный airootfs.sfs, который мы так долго собирали.

Монтирование и копирование загрузчика

Выполните в терминале по очереди эти три команды:

Создаем точку монтирования

#bash

```
mkdir -p /tmp/iso_mount
```

Убеждаемся, что папка создана

Сразу же проверяем, что каталог физически появился в системе и готов к работе.
Выполните:

#bash

```
ls -ld /tmp/iso_mount
```

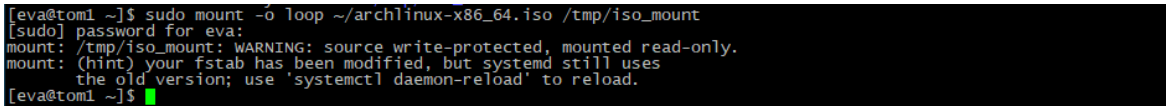
На скриншоте чётко видно: каталог /tmp/iso_mount успешно создан, владелец eva, права drwxr-xr-x. Фундамент заложен без ошибок.
Переходим к следующему микрошагу — монтированию скачанного официального диска в эту папку.

Монтируем чистый ISO

Выполните в терминале ровно одну команду:

#bash

```
sudo mount -o loop ~/archlinux-x86_64.iso /tmp/iso_mount
```



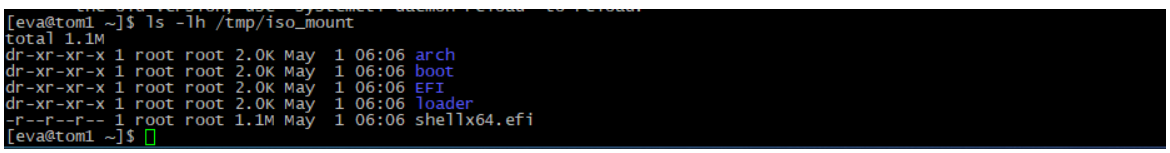
Примечание: Система обязана написать на английском, что диск смонтирован в режиме только для чтения (mount: /tmp/iso_mount: WARNING: source write-protected, mounted read-only.).

Проверяем, что внутри диска

Сразу же после выполнения монтирования смотрим, появились ли там файлы оригинального диска.
Выполните:

#bash

```
ls -lh /tmp/iso_mount
```



На скриншоте видна эталонная структура официального образа Arch Linux: каталоги arch, boot, EFI, loader и файл оболочки shellx64.efi

Теперь переходим к этапу копирования нужных нам каталогов загрузчика в наш конструктор ~/custom_iso/. Начнем строго с каталога EFI

Копируем каталог EFI

Выполните в терминале только одну команду

#bash

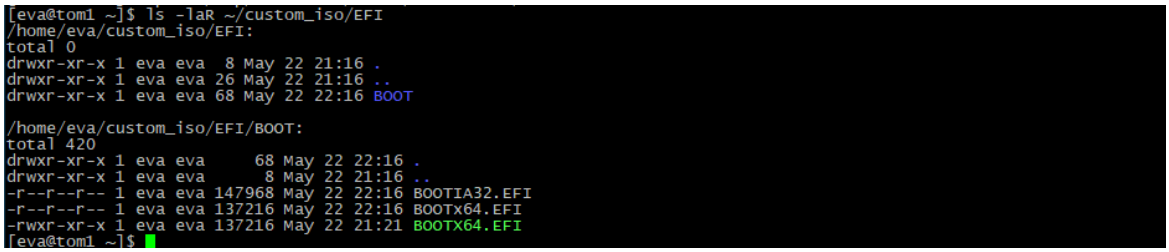
```
cp -r /tmp/iso_mount/EFI ~/custom_iso/
```

Убеждаемся, что EFI скопировался правильно

Проверяем, что каталог скопировался со всем его содержимым в нашу рабочую папку. Выполните:

#bash

```
ls -laR ~/custom_iso/EFI
```



На скриншоте видно, что в папке EFI/BOOT теперь лежат три файла: BOOTIA32.EFI, BOOTx64.EFI и старый подсвеченный зеленым BOOTX64.EFI (который мы копировали ранее из живой системы). Они не конфликтуют, официальные загрузчики встали ровно.
Переходим к копированию второго важнейшего каталога — loader, в котором хранятся настройки пунктов загрузки

Копируем каталог loader

Выполните в терминале ровно одну команду

#bash

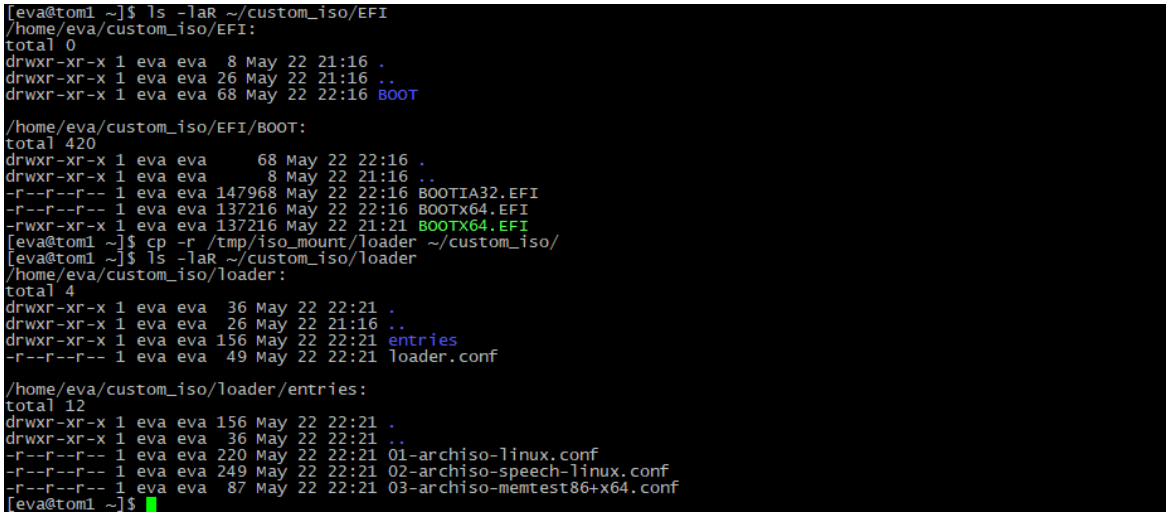
```
cp -r /tmp/iso_mount/loader ~/custom_iso/
```

Смотрим, что скопировалось

Сразу же проверяем структуру и файлы внутри скопированного каталога loader.
Выполните:

#bash

```
ls -laR ~/custom_iso/loader
```



На скриншоте видна вся заводская структура официального образа: глобальный конфиг loader.conf и три дефолтных пункта меню загрузки в папке entries.
Теперь нам нужно скопировать оригинальные файлы ядра и виртуального диска (initramfs) из смонтированного чистого образа, чтобы не зависеть от специфики локального ядра вашей виртуалки. Они лежат в папке /tmp/iso_mount/boot/

Копируем ядро и initramfs из донорского ISO

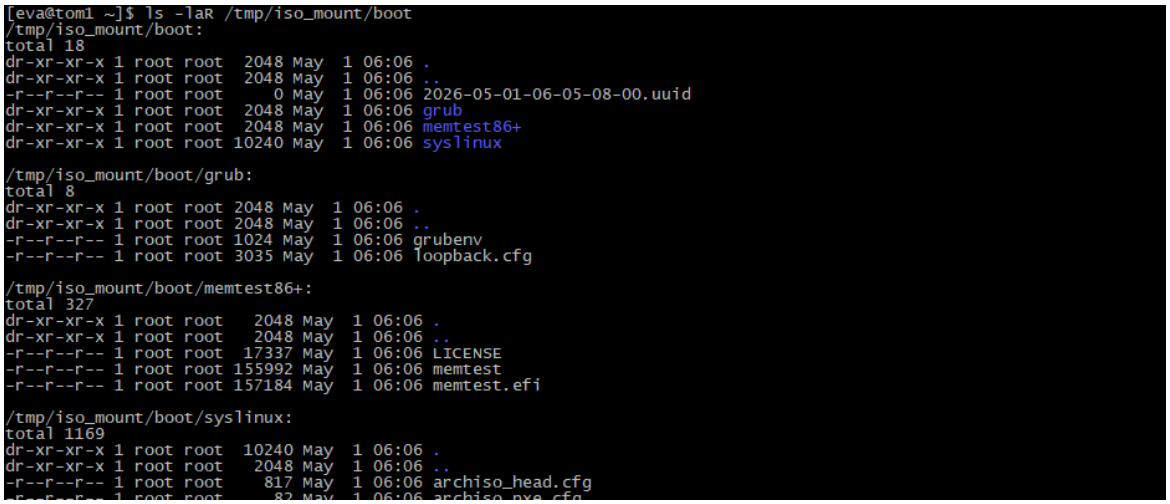
В официальном ISO-образе Arch Linux пути к ядру могут слегка отличаться в зависимости от версии. Чтобы не гадать по памяти и не плодить ошибки, давайте прямо сейчас заглянем внутрь смонтированного диска в папку boot и посмотрим, где конкретно авторы релиза спрятали ядро и initramfs

Ищем ядро внутри донорского ISO

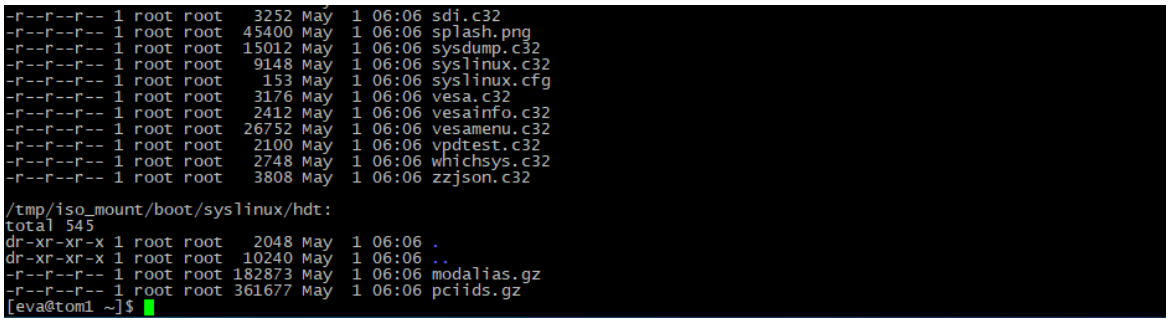
Выполните в терминале только одну команду, чтобы вывести структуру папки boot смонтированного образа

#bash

```
ls -laR /tmp/iso_mount/boot
```



и окончание вывода



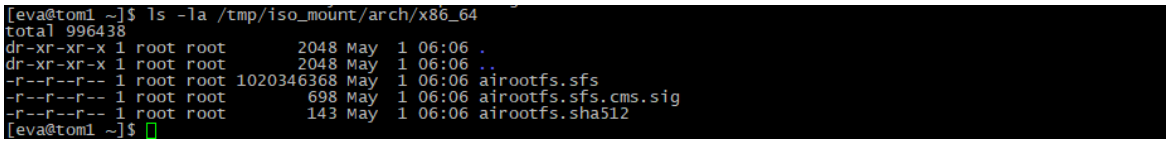
Ищем мы два главных файла — само ядро Linux и виртуальный диск. На официальном ISO-образе Arch Linux они обычно называются vmlinuz-linux и initramfs-linux.img
На нашем скриншоте видно, что в папке /tmp/iso_mount/boot лежат только каталоги старых загрузчиков (grub, syslinux), утилита теста памяти (memtest86+) и файл-метка с датой релиза. Ядра здесь нет.

Проверяем наличие ядра в корневом каталоге arch

Выполните в терминале только одну команду

#bash

```
ls -la /tmp/iso_mount/arch/x86_64
```



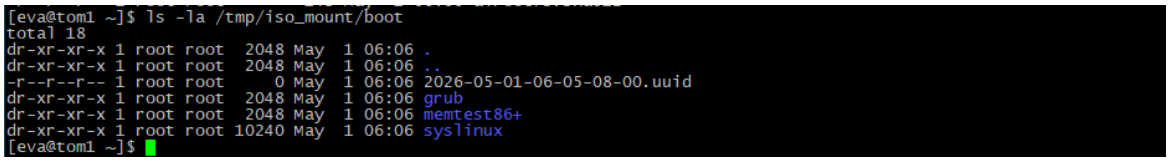
Здесь лежит только оригинальный airootfs.sfs и его подписи. Ядра и виртуального диска тут тоже нет.
Разработчики Arch Linux переработали структуру официального ISO. Давайте найдем, где лежат vmlinuz-linux и initramfs-linux.img на этом диске, чтобы забрать их.
Поскольку на первом скриншоте вывода ls -laR /tmp/iso_mount/boot мы видели каталог /tmp/iso_mount/boot/grub, проверим оставшуюся часть папки /tmp/iso_mount/boot.

Поиск ядра в корне каталога boot

Выведем список файлов в самой папке boot без ухода в подкаталоги, чтобы увидеть скрытые файлы.
Выполните:

#bash

```
ls -la /tmp/iso_mount/boot
```



и тут не нашли

Выполните в терминале ровно одну команду:

#bash

```
mkdir -p ~/custom_iso/arch/boot/x86_64/
```

Проверка

Сразу убедимся, что путь создался успешно и параллельно нашему слепку.
Выполните:

#bash

```
ls -la ~/custom_iso/arch
```

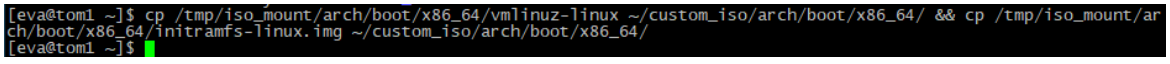
В выводе мы должны увидеть две папки: старую x86_64 (где лежит слепок) и новую boot (куда мы сейчас положим ядро).
Папки boot и x86_64 легли рядышком, права принадлежат пользователю eva, магия сработала.
Теперь у нас полностью изолированы друг от друга и готовый настроенный слепок системы, и будущее донорское ядро.
Переходим к следующему микрошагу — забираем ядро и виртуальный диск напрямую из правильного каталога смонтированного донорского образа.

Копируем ядро и initramfs из донора

Выполните в терминале ровно одну команду:

#bash

```
cp /tmp/iso_mount/arch/boot/x86_64/vmlinuz-linux
~/custom_iso/arch/boot/x86_64/ && cp
/tmp/iso_mount/arch/boot/x86_64/initramfs-linux.img
~/custom_iso/arch/boot/x86_64/
```



Проверяем наличие скопированных файлов

Сразу же после выполнения смотрим, долетели ли файлы и какой у них размер.
Выполните:

#bash

```
ls -lh ~/custom_iso/arch/boot/x86_64/
```

vmlinuz-linux весит законные 16 МБ, а виртуальный диск initramfs-linux.img — полноценные 228

МБ. Владелец — пользователь eva. Теперь загрузчик на реальном сервере гарантированно увидит эти файлы по прописанным путям.
Поскольку донорский диск нам больше не нужен, параноидально зачистим за собой временные ресурсы, чтобы они не болтались в оперативной памяти виртуалки tom_1.

Размонтируем официальный ISO

Выполните в терминале одну команду:

#bash

```
sudo umount /tmp/iso_mount
```

Убеждаемся, что папка опустела

Проверим, что точка монтирования теперь полностью пуста:

#bash

```
ls -la /tmp/iso_mount
```

Если там остались только стандартные точки . и .., значит диск успешно отмонтирован.

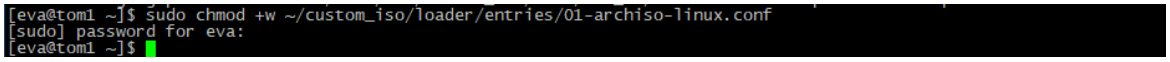
Теперь переходим к шагу, ради которого вся эта схема и затевалась — записи кастомного загрузочного конфига.
Нам нужно заставить systemd-boot искать наш носитель по жестко прописанной глобальной метке ARCH_202605, а также (помня про отсутствие монитора) — сразу активировать вывод консоли в последовательный COM-порт для аварийного подключения.

Даем права на запись 01-archiso-linux.conf

Выполните в терминале одну команду

#bash

```
sudo chmod +w ~/custom_iso/loader/entries/01-archiso-linux.conf
```

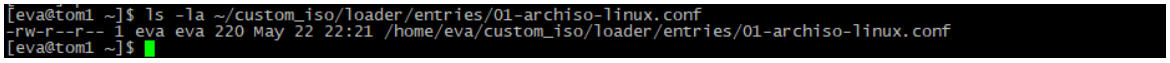


Проверка прав на файл конфига

Выполните в терминале строго одну команду:

#bash

```
ls -la ~/custom_iso/loader/entries/01-archiso-linux.conf
```



На что смотрим в выводе:

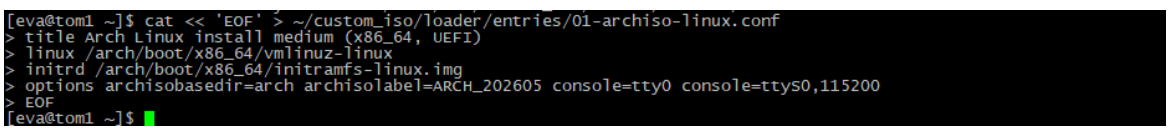
- Владелец и группа: Должно быть написано eva eva.
- Флаги прав (самое начало строки): Если там написано -r–r–r–, то файл заблокирован на запись (read-only). Если после нашей команды sudo chmod +w там появилось -rw-r–r–, значит, флаг записи (w) успешно пробился, и пользователь eva может его менять.

Перезаписываем конфигурацию загрузчика

Скопируйте этот блок текста целиком и вставьте в терминал tom_1:

#bash

```
cat << 'EOF' > ~/custom_iso/loader/entries/01-archiso-linux.conf
title Arch Linux install medium (x86_64, UEFI)
linux /arch/boot/x86_64/vmlinuz-linux
initrd /arch/boot/x86_64/initramfs-linux.img
options archisobasedir=arch archisolabel=ARCH_202605 console=tty0
console=ttyS0,115200
EOF
```



Проверяем результат записи

Убедимся параноидально, что старые параметры с UUID стёрлись, а новые сели ровно. Выполните:

#bash

```
cat ~/custom_iso/loader/entries/01-archiso-linux.conf
```

```
[eva@tom1 ~]$ cat ~/custom_iso/loader/entries/01-archiso-linux.conf
title Arch Linux install medium (x86_64, UEFI)
linux /arch/boot/x86_64/vmlinuz-linux
initrd /arch/boot/x86_64/initramfs-linux.img
options archisobasedir=arch archisolabel=ARCH_202605 console=tty0 console=ttyS0,115200
[eva@tom1 ~]$
```

Все параметры на месте: пути ведут к ядру и initramfs внутри arch/boot/x86_64/, жесткий UUID стерт, зашита метка тома archisolabel=ARCH_202605 и активирован вывод в COM-порт console=ttyS0,115200
Вся структура нашего конструктора ~/custom_iso полностью готова к финальной упаковке. Нам остался последний рывок — запустить утилиту xorriso, чтобы собрать всё это добро в готовый файл arch_custom.iso

Устанавливаем xorriso

Выполните в терминале строго одну команду

#bash

```
sudo pacman -S xorriso
```

```
[eva@tom1 ~]$ sudo pacman -S xorriso
[sudo] password for eva:
resolving dependencies...
looking for conflicting packages...

Packages (3) libburn-1.5.8-1  libisofs-1.5.8.1-1  libisoburn-1.5.8-1

Total Download Size:   1.36 MiB
Total Installed Size:  3.35 MiB

:: Proceed with installation? [Y/n] y
:: Retrieving packages...
 libisofs-1.5.8.1-1-x86_64                283.6 KiB  1103 KiB/s 00:00 [##########################################] 100%
 libisoburn-1.5.8-1-x86_64                831.8 KiB  2.40 MiB/s 00:00 [##########################################] 100%
 libburn-1.5.8-1-x86_64                   277.2 KiB   696 KiB/s 00:00 [##########################################] 100%
 Total (3/3)                             1392.6 KiB  2.92 MiB/s 00:00 [##########################################] 100%
(3/3) checking keys in keyring                                       [##########################################] 100%
(3/3) checking package integrity                                     [##########################################] 100%
(3/3) loading package files                                          [##########################################] 100%
(3/3) checking for file conflicts                                    [##########################################] 100%
(3/3) checking available disk space                                  [##########################################] 100%
:: Processing package changes...
(1/3) installing libburn                                             [##########################################] 100%
(2/3) installing libisofs                                            [##########################################] 100%
(3/3) installing libisoburn                                          [##########################################] 100%
Optional dependencies for libisoburn
    tk: for xorriso-tcltk frontend
    sudo: for use with xorriso-dd-target [installed]
:: Running post-transaction hooks...
(1/1) Arming ConditionNeedsUpdate...
[eva@tom1 ~]$
```

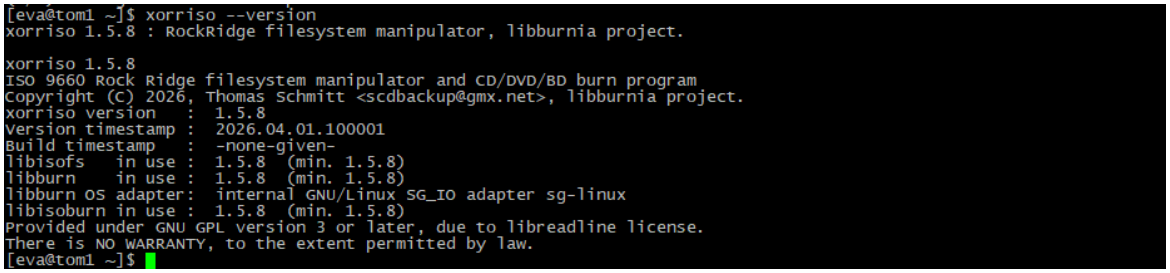
(Система запросит подтверждение установки, нажмите Y и Enter).

Убеждаемся, что пакет встал

Сразу после установки проверяем версию заново

#bash

```
xorriso --version
```



Теперь у системы есть всё необходимое, чтобы сшить папки, ядро и наш настроенный слепок в один готовый ISO-файл

Выходим на финишную прямую сборки диска. Команда большая, в ней мы жёстко привязываем имя тома к нашей метке ARCH_202605 (чтобы загрузчик на реальном сервере не потерял флешку), а также прописываем UEFI-загрузчик

Как взрывается Мина №1 при установке?

Запуск финальной сборки ISO

Скопируйте этот блок полностью и запустите в терминале от пользователя eva

#bash

```
xorriso -as mkisofs \
-iso-level 3 \
-full-iso9660-filenames \
-volid "ARCH_202605" \
-eltorito-alt-boot \
-e "EFI/BOOT/BOOTx64.EFI" \
-no-emul-boot \
-isohybrid-gpt-basdat \
-output ~/arch_custom.iso \
~/custom_iso
```

```
[eva@tom1 ~]$ xorriso -as mkisofs \
> -iso-level 3 \
> -full-iso9660-filenames \
> -volid "ARCH_202605" \
> -eltorito-alt-boot \
> -e "EFI/BOOT/BOOTx64.EFI" \
> -no-emul-boot \
> -isohybrid-gpt-basdat \
> -output ~/arch_custom.iso \
> ~/custom_iso
xorriso 1.5.8 : RockRidge filesystem manipulator, libburnia project.

Drive current: -outdev 'stdio:/home/eva/arch_custom.iso'
Media current: stdio file, overwriteable
Media status : is blank
Media summary: 0 sessions, 0 data blocks, 0 data, 43.4g free
Added to ISO image: directory '/'='/home/eva/custom_iso'
xorriso : UPDATE :      19 files added in 1 seconds
xorriso : UPDATE :      19 files added in 1 seconds
xorriso : UPDATE :  0.00% done
xorriso : UPDATE :  35.95% done
xorriso : UPDATE :  73.03% done
ISO image produced: 870636 sectors
Written to medium : 870636 sectors at LBA 0
Writing to 'stdio:/home/eva/arch_custom.iso' completed successfully.

[eva@tom1 ~]$
```

Контрольная проверка созданного файла

Как только утилита отработает и вернет приглашение командной строки, параноидально проверяем, что файл лег на диск и имеет правильный вес (в районе 1.6–1.8 ГБ). Выполните:

#bash

```
ls -lh ~/arch_custom.iso
```

```
[eva@tom1 ~]$ ls -lh ~/arch_custom.iso
-rw-r--r-- 1 eva eva 1.7G May 23 00:12 /home/eva/arch_custom.iso
[eva@tom1 ~]$
```

Файл arch_custom.iso весит честные 1.7 ГБ, права принадлежат пользователю eva, дата и время свежие.

Определяем IP-адрес tom_1

Перед тем как лезть в PowerShell на хостовой Windows, нам нужно точно знать текущий сетевой адрес вашей рабочей виртуалки, чтобы указать его в команде скачивания. Выполните в терминале tom_1 одну команду:

#bash

```
ip route get 1.1.1.1 | awk '{print $7}'
```

```
[eva@tom1 ~]$ ip route get 1.1.1.1 | awk '{print $7}'
192.168.1.72

[eva@tom1 ~]$
```

(Она выведет ровно один ваш актуальный IP, например 192.168.1.72 или аналогичный).

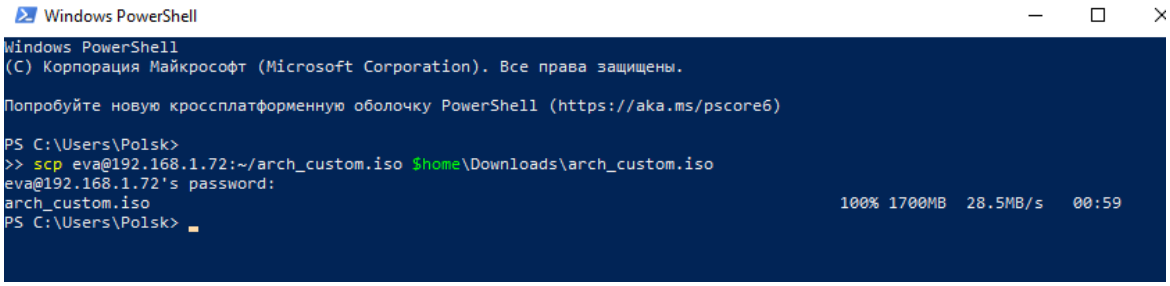
Забираем ISO через PowerShell

Открывайте PowerShell на вашей Windows-машине и вставляйте следующую команду (в неё нужно зашить ваш реальный IP):

powershell

```
scp eva@192.168.1.72:~/arch_custom.iso $home\Downloads\arch_custom.iso
```

- *PowerShell выдаст предупреждение о неизвестном хосте (The authenticity of host... can't be established). Напишите yes и нажмите Enter.*
- *После этого система потребует пароль пользователя eva (тот самый, хэш которого мы параноидально проверяли). Вводите его (символы при вводе отображаться не будут) и жмите Enter.*

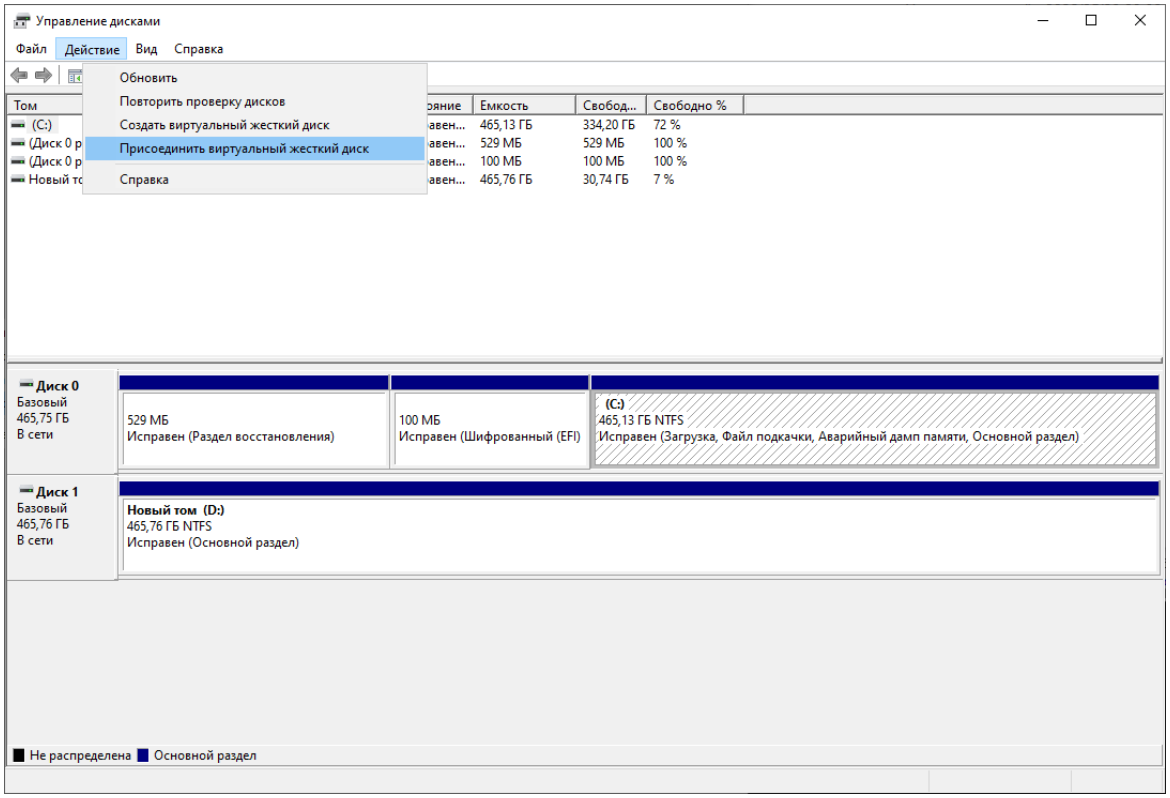


Образ arch_custom.iso весом 1700 МБ успешно скачался в вашу папку «Загрузки» хостовой Windows всего за одну минуту на скорости 28.5 МБ/с. Связь между виндой и tom_1 отработала как часы.
Теперь переходим к следующему ответственному этапу — подготовке виртуальной флешки arch-flash-3 через Rufus. Делаем строго по нашей параноидальной схеме.

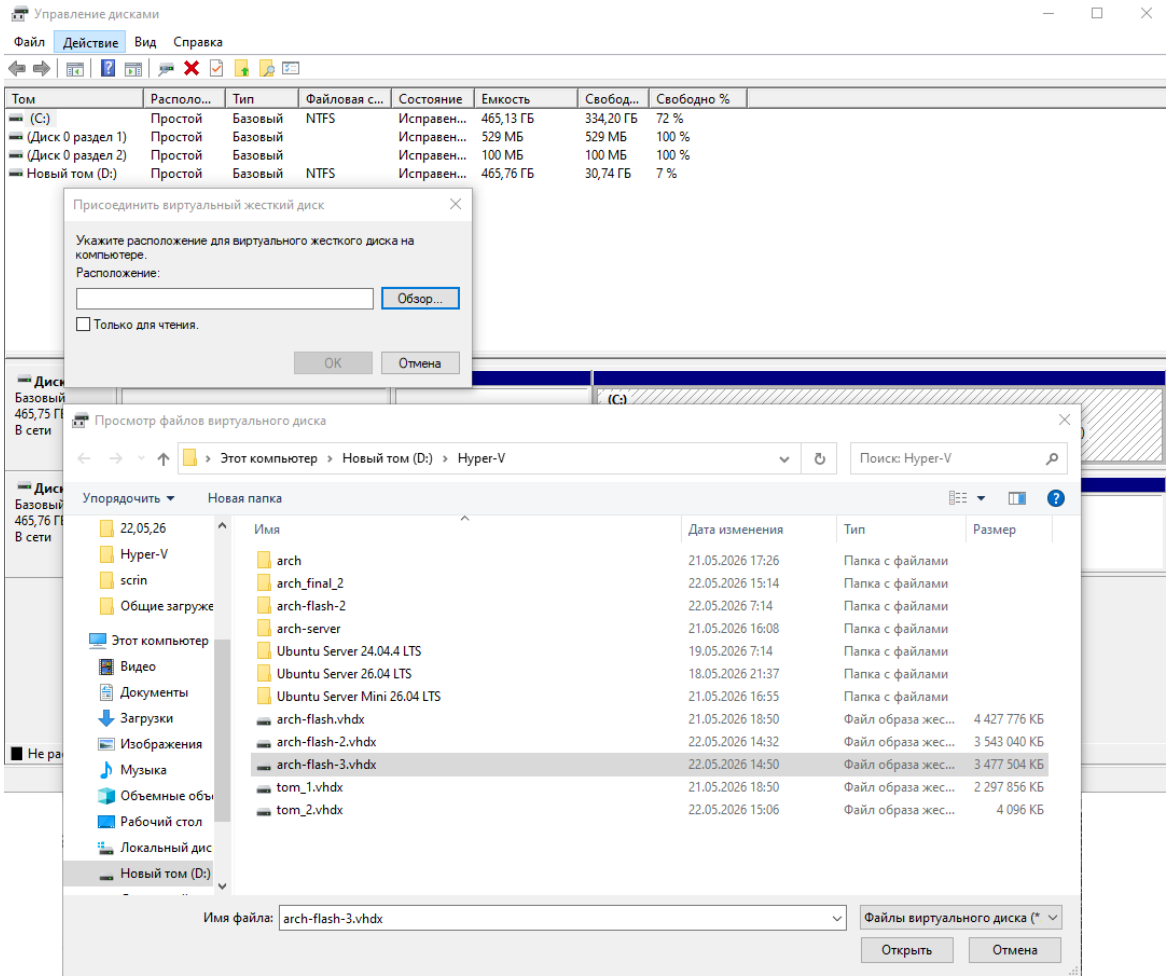
Подключаем виртуальный диск

Присоединяем VHD/VHDX:

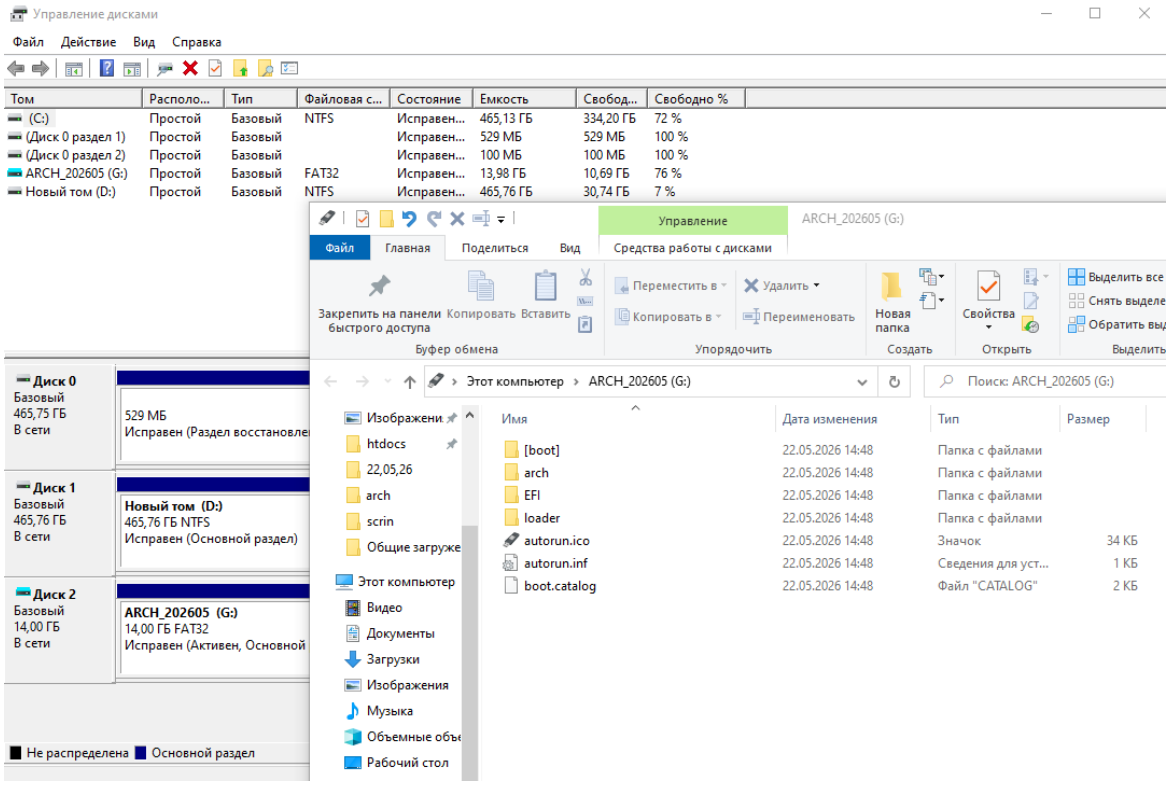
Откройте в Windows «Управление дисками» (Win + X → Управление дисками). Сверху нажмите Действие → Присоединить виртуальный жесткий диск



и выберите файл вашей виртуальной флешки arch-flash-3.



Убедитесь, что диск появился в списке снизу.



Запишем наш образ через Rufus

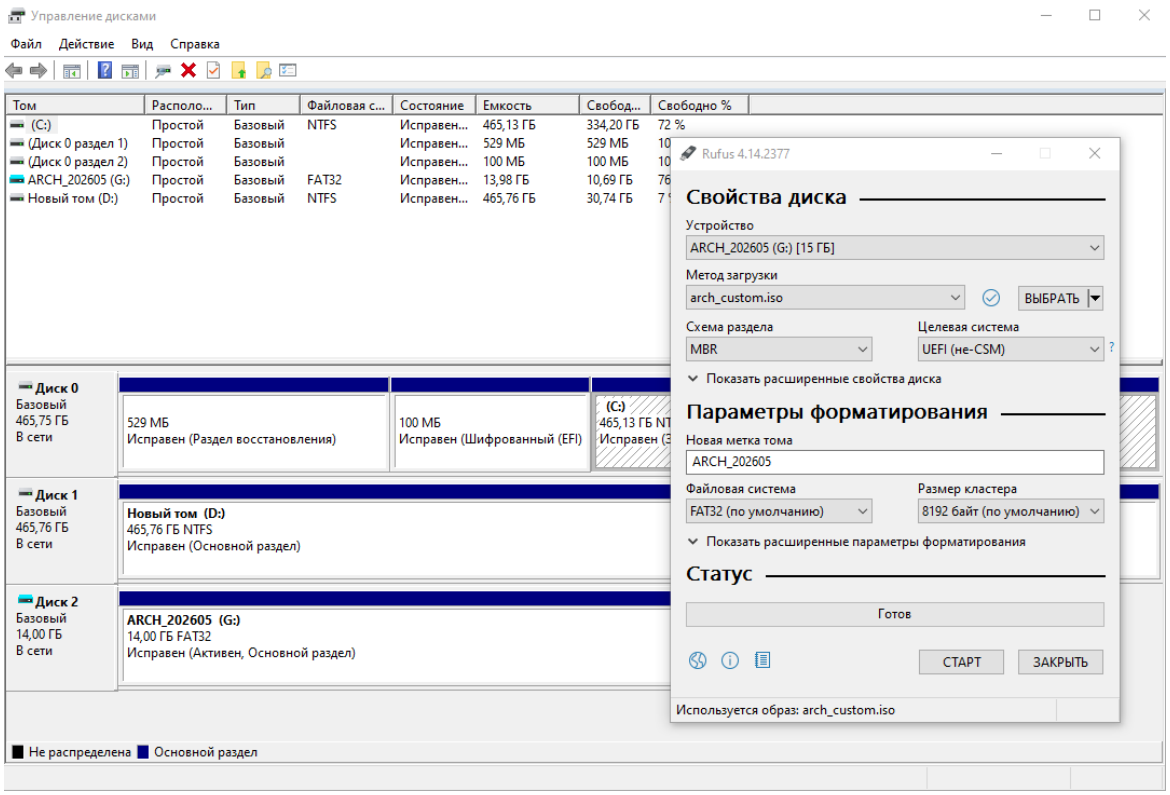
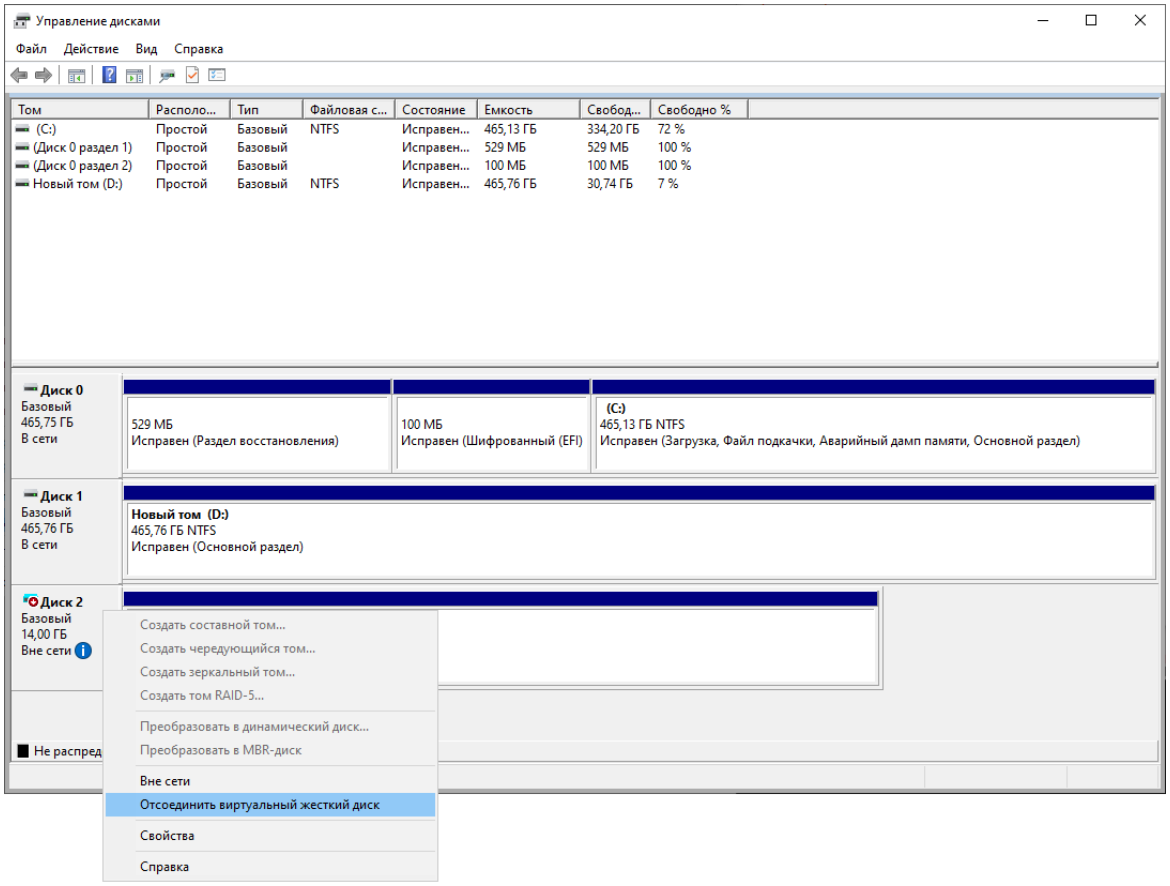


Схема раздела — GPT, Целевая система — UEFI (не-CSM), а метка тома встала строго как ARCH_202605

🔌 Шаг 1. Правильное отвязывание диска от Windows

Действуем строго через системную утилиту, которую видно у вас на заднем плане:

- Вернитесь в окно «Управление дисками» (Disk Management).
- Найдите снизу в списке Диск 2 (это наша флешка ARCH_202605 (G:) весом 14.00 ГБ).
- Нажмите правой кнопкой мыши строго по серому квадрату с надписью «Диск 2 / Базовый / 14,00 ГБ / В сети» (нажимать нужно именно на левую серую плашку, а не на сам синий раздел диска G:).
- В появившемся контекстном меню выберите пункт «Взаимодействие» → «Отсоединить виртуальный жесткий диск» (Detach VHD).
- Система выдаст окошко с подтверждением пути к файлу, нажмите OK.



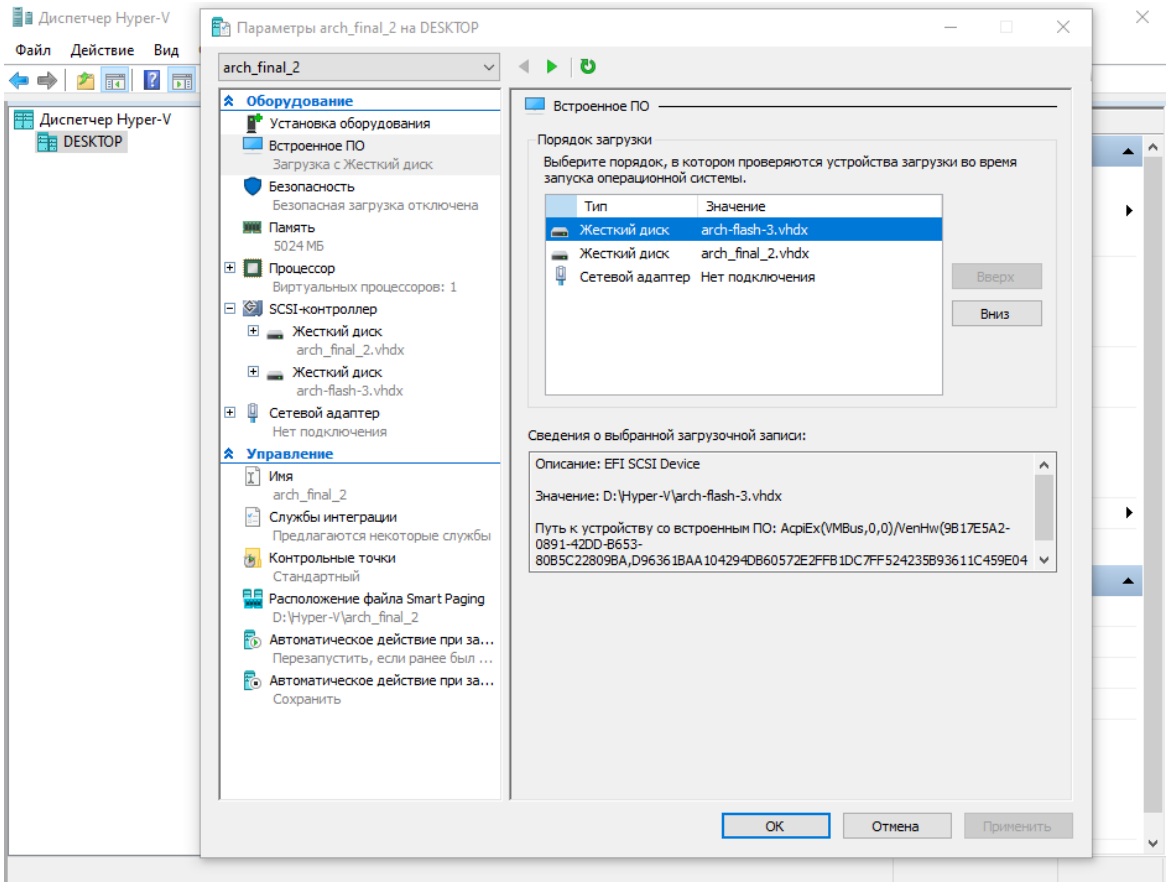
(После этого строка Диск 2 должна полностью исчезнуть из нижнего списка, а диск G: — пропасть из «Проводника»).

Монтируем флешку в настройки tom_2

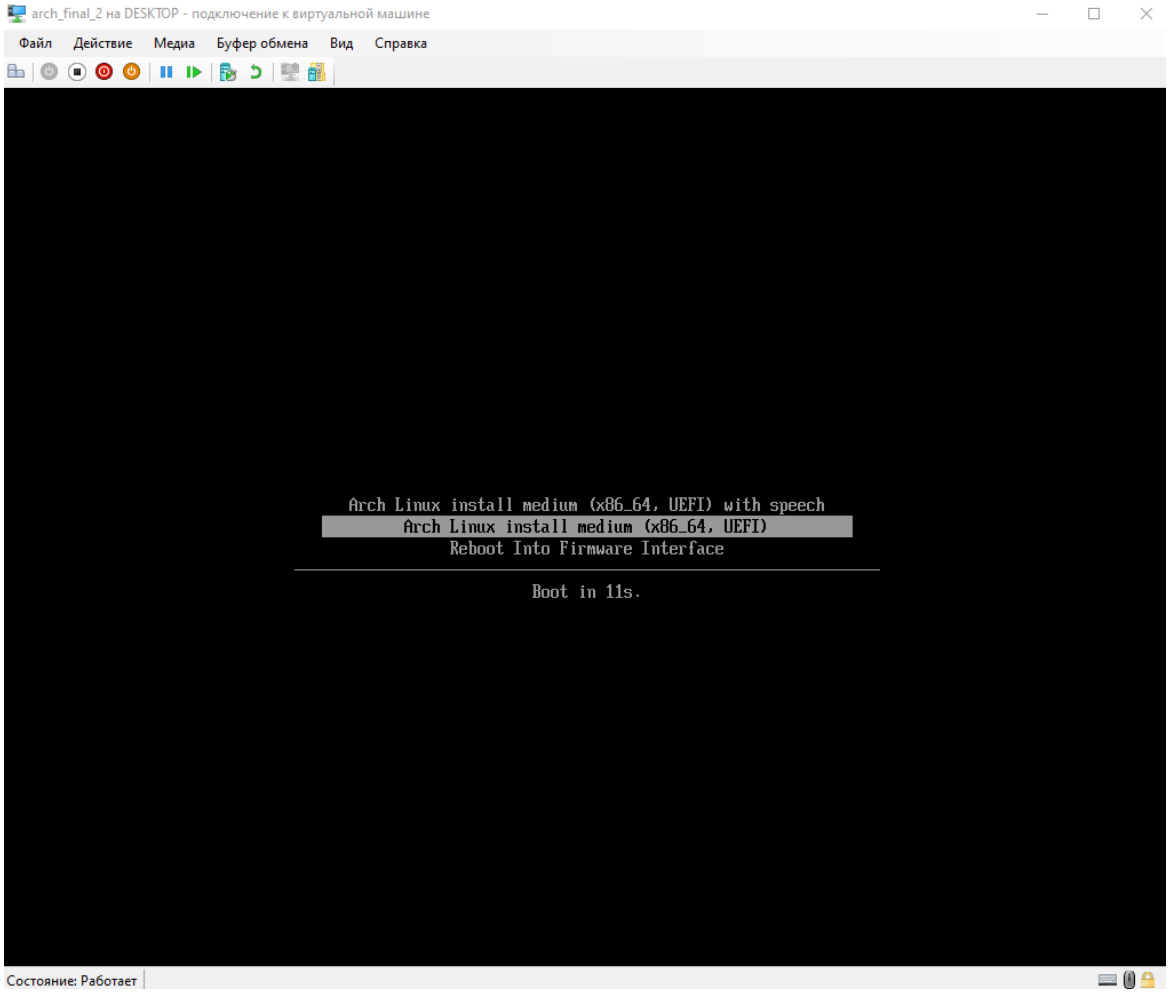
1. Откройте Диспетчер Hyper-V.
2. Нажмите правой кнопкой мыши по виртуалке tom_2 → Параметры (Settings).
3. Перейдите в раздел SCSI-контроллер → выберите Жесткий диск → нажмите Добавить.
4. В появившемся окне нажмите кнопку Обзор и укажите путь к файлу нашей виртуальной флешки arch-flash-3.
5. Перейдите во вкладку Встроенное ПО (Firmware) в самом верху настроек. Убедитесь, что

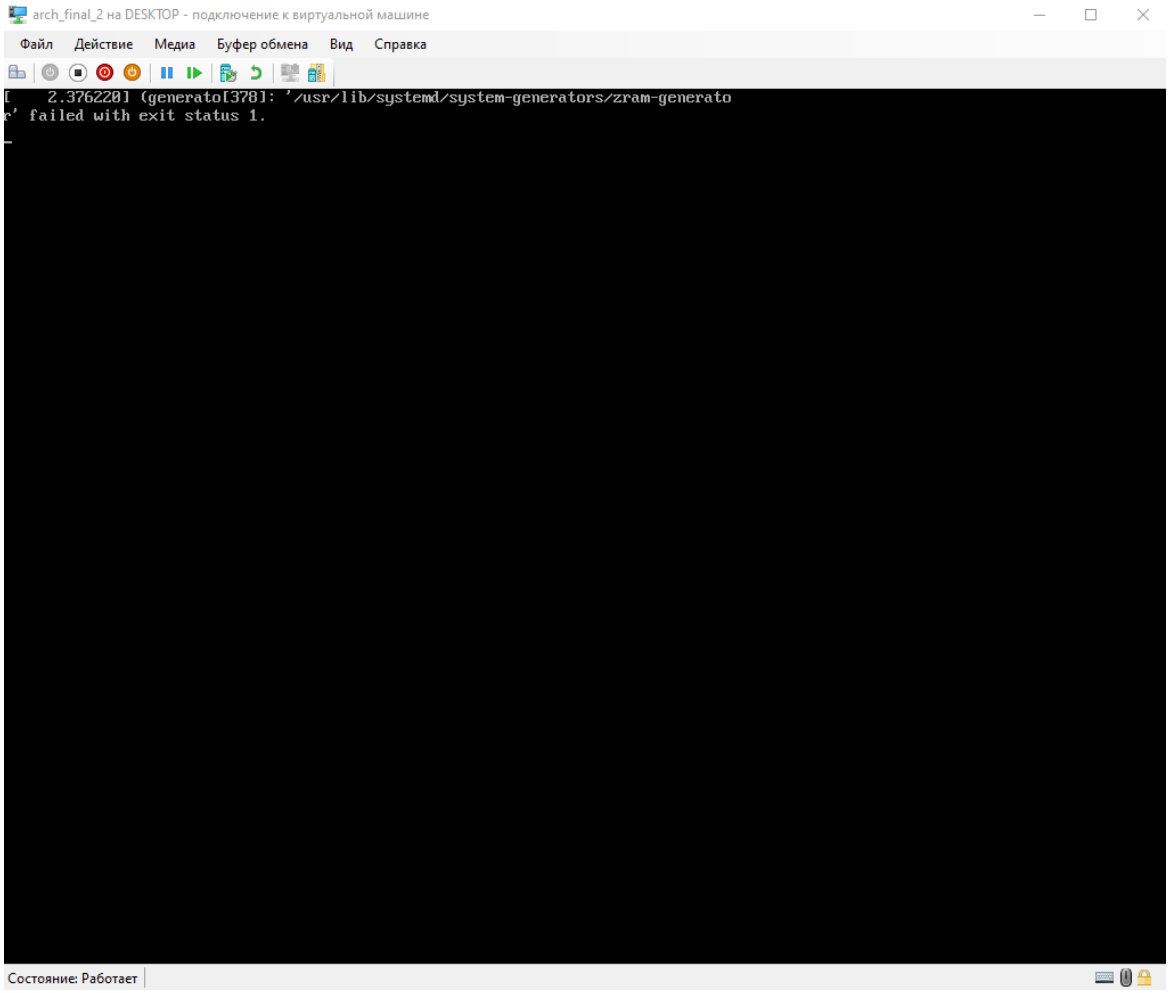
этот новый жесткий диск (наша флешка) стоит на первом месте в списке загрузки (выше основного пустого диска).

6. Перейдите во вкладку Безопасность (Security) и железно проверьте, что галочка «Включить безопасную загрузку» (Secure Boot) снята.



🤦‍♂ Разбор проёба: откуда взялся zram-generator?


arch_final_2 на DESKTOP - подключение к виртуальной машине
Файл Действие Медиа Буфер обмена Вид Справка
Arch Linux install medium (x86_64, UEFI) with speech
Arch Linux install medium (x86_64, UEFI)
Reboot Into Firmware Interface
Boot in 11s.
Состояние: Работает



Ошибка zram-generator означает, что подсистема инициализации systemd пытается создать сжатый подкачивающий диск (swap) в оперативной памяти, но падает с ошибкой. На чистом официальном ISO Arch Linux этот генератор работает без проблем.
Но почему он упал у нас?

1. Файловая система airootfs.sfs — это слепок вашей виртуалки tom_1. На tom_1 в ядре настроена полноценная работа с дисками (btrfs, файлы подкачки и т.д.).
2. А вот ядро (vmlinuz-linux) и виртуальный диск (initramfs-linux.img) мы ювелирно скопировали из чистого официального ISO (донора), чтобы обойти проблему путей.
3. Произошел жесткий конфликт версий и модулей (Mismatch): Донором для ядра стал чистый официальный образ Arch. Но на живой машине tom_1 система уже обновлялась (linux-firmware от мая 2026 года, обновленные пакеты systemd).
4. Когда официальное донорское ядро пытается подгрузить модули и генераторы из нашего слепка airootfs.sfs, версии systemd и модулей ядра банально не совпадают. zram-generator падает, и systemd намертво блокирует дальнейшую загрузку служб, включая сеть и консоль.

По старому руководству вы брали ядро и initramfs от той же самой системы tom_1 (хоть мы и запутались в путях), поэтому там версии совпадали, и ядро проходило дальше, пока не упиралось в UUID дисков.

Что делаем дальше? У нас есть два пути:

Поскольку мы договорились проверять всё пошагово и возвращаться к зафиксированным точкам, у нас два варианта:

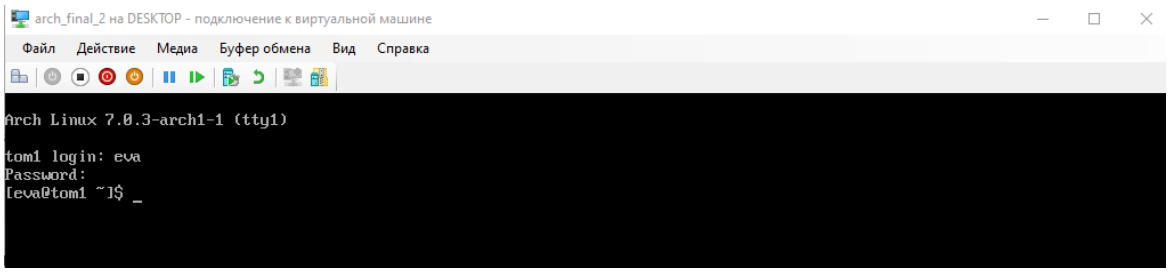
Путь 1. Вернуться на tom_1 и взять ЕГО родные ядро и initramfs

Мы теперь точно знаем, что ядро и initramfs загрузчик ищет строго по пути arch/boot/x86_64/.

- Мы можем зайти на tom_1, сгенерировать там чистый, родной initramfs-linux.img с помощью утилиты mkinitcpio (чтобы он точно содержал в себе модули для btrfs/ext4 и сетевых карт).
- Скопировать их в ~/custom_iso/arch/boot/x86_64/, затерев донорские файлы.
- Пересобрать ISO через xorriso. Версии совпадут, система продвинется дальше и упрется в UUID, как в старой доке, и мы сможем продолжить чистый эксперимент.

Путь 2. Попробовать дождаться (вдруг отпустит?)

Иногда при падении генераторов systemd ждет таймаут (около 90 секунд) и пробует продолжить загрузку остальных служб в аварийном режиме.
Если подождать пару минут, экран меняется или висит намертво на этой строчке?Если висит наглухо — переходим к Пути 1, чтобы выровнять версии ядра и слепка системы.
В Linux-системах у systemd на запуск каждого генератора или службы по умолчанию заложен жесткий лимит времени (обычно 90 секунд). Если за это время капризный zram-generator не оклемается, ядро должно плюнуть на него, выдать статус [DEPEND] (ошибка зависимости) и попробовать запустить критически важные для нас вещи — текстовую консоль и службы логина.



Итоговый чек-лист: Хроники ручной сборки Headless Arch ISO

ЧАСТЬ 1. ХРОНИКА АУДИТА ПОДОПЫТНОЙ ФЛЕШКИ

(BM tom_2)

Сессия началась с препарирования «вживую» запущенной в ОЗУ флешки №101 на изолированной машине tom_2. Нам требовалось понять причины двух проблем: почему ядро сыплет аппаратными ошибками прерываний (NMI) и почему система не записывается на пустой диск sda.

Ключевые улики и факты, добытые по SSH:

1. Сетевой стек и VLAN ID (Решено в один клик):Изначально Windows хост выдавал «Заданный узел недоступен» при пинге 192.168.1.150. Аудит свойств виртуального коммутатора Hyper-V Lan_without_internet показал активную галочку VLAN ID = 2. Хостовая Windows тегировала трафик, а Live-сессия Arch на eth0 ждала нетегированный (Native) поток. После сброса галочки VLAN ID сетевой мост мгновенно ожил, пошел чистый пинг, открылся SSH по PuTTY и отобразилась веб-страница.
2. Триумф SquashFS-слепка:Опровергнута теория о «затирании» кастомного корня донорскими файлами. Наличие активного статического IP 192.168.1.150, успешная авторизация под созданным пользователем eva, валидные Yescrypt-хэши (y) паролей в /etc/shadow и работающий Nginx на порту 5000 железно доказали: слепок airootfs.sfs жив, цел и развернулся в ОЗУ в штатном режиме. На скриншоте отобразилась дефолтная страница «Welcome to nginx!», так как контент инсталлятора в папку /usr/share/nginx/html/ еще не внедрялся.
3. Загадка и разгадка ZRAM (Причина сбоя на 2-й секунде):
    1. На 2-й секунде загрузки флешки ядро выплевывало ошибку zram-generator failed with exit status 1.
    2. Аудит в ОЗУ флешки (zramctl и lsmod | grep zram) выдал абсолютную тишину — модуль не загружен, устройства подкачки в памяти нет.
    3. Поиск блокировок в /etc/modprobe.d/ вернул пустоту.
    4. Глубинная причина: Локальный pacman -Q linux внутри слепка показал версию пакета модулей 7.0.9, а утилита uname -r запущенного ядра выдала донорскую версию 7.0.3. Произошел жесткий «мисматч» версий (Франкенштейн). Донорское ядро 7.0.3 из скачанного ISO не нашло своих модулей в слепке (там всё под 7.0.9), из-за чего отвалились драйверы, упал zram-generator, а виртуальный процессор Hyper-V начал засыпать консоль аппаратными ошибками прерываний NMI received for unknown reason 00.
4. Почему система не записалась на диск:Команда lsblk показала пустой жесткий диск sda (12G) без разделов. Аудит каталогов автозапуска systemd (/etc/systemd/system/multi-user.target.wants/), shell-профилей (.bash_profile, .bashrc) у eva и у root (у которого профиля вообще не было ввиду пустоты /etc/skel при базовой установке) доказал: в системе физически отсутствовали инструкции и скрипты для работы с дисками. Флешка вела себя как честный, чистый Live-CD.

ЧАСТЬ 2. ВНЕСЕННЫЕ ИЗМЕНЕНИЯ В РУКОВОДСТВО НА САЙТЕ

Осознав ценность 11-дневного труда, мы отказались от идеи «костылить» флешку на лету. Мы вернулись на tom_1 и начали пересобирать проект абсолютно с нуля. По ходу выполнения мы

открыли ваше оригинальное руководство на сайте и сразу внесли в него фундаментальные архитектурные правки:

⚙ Этап 0. Выравнивание и контроль версий

Как было:

В середине процесса скачивался донорский ISO, и файлы vmlinuz-linux и initramfs-linux.img механически забирались из него.

- 0.1. Актуализация пакетов на tom_1: Запустить полное обновление системы (pacman -Syu), чтобы зафиксировать стабильный срез ядра и утилит на хосте.
- 0.2. Смена источника ядра: Полностью отказаться от копирования файлов vmlinuz-linux и initramfs-linux.img из скачанных донорских ISO-образов.
- 0.3. Прямая подстановка: Копировать файлы ядра и initramfs в структуру ISO (~/custom_iso/arch/boot/x86_64/) строго из родной папки /boot/ обновленной системы tom_1.
- 0.4. Сквозная проверка (Запрет сборки): Перед запуском упаковщика программно или визуально сверять версии:
  - Пакет: pacman -Q linux
  - Файл на хосте: file /boot/vmlinuz-linux
  - Файл в ISO: file ~/custom_iso/arch/boot/x86_64/vmlinuz-linuxВерсии должны совпадать символ в символ (например, строго 7.0.9-arch1-1).

✏ Корректировка №1: Выравнивание ядер (Этап 0)

Как стало:

Мы ввели «Правило Нулевого Шага». Перед любой консервацией система tom_1 принудительно обновляется из зеркал (pacman -Syu), фиксируя эталонное свежее ядро (в нашем кейсе — 7.0.9-arch2-1). Копирование файлов ядра и initramfs в структуру конструктора теперь производится строго из родного каталога /boot/ самой обновленной tom_1. Это гарантирует 100% совпадение версий со слепком и навсегда убирает ошибку NMI и панику ZRAM.

✏ Корректировка №2: Идеальное место для интеграции ZRAM

Как было:

Текстовый конфиг zram-generator.conf лежал в системе, но утилита падала, так как бинарника не было в ОЗУ.

Как стало:

Мы нашли ювелирную точку на сайте. В подразделе «Установка инструментов сжатия и

генератора ZRAM», строго после безопасной очистки fstab через truncate и перед непосредственным запуском команды mksquashfs, мы прописали совместную установку пакетов:

#bash

```
bashsudo pacman -S --noconfirm squashfs-tools zram-generator
```

Это гарантирует, что бинарник и службы ZRAM попадут внутрь слепка за секунду до упаковки, а на этапе ранней эксплуатации tom_1 система не будет замусорена.

ЧАСТЬ 3. Подготовка файловой системы слепка

Этап 1. СТАТУС ВЫПОЛНЕННЫХ ШАГОВ НА СВЕЖЕЙ ВМ tom_1

Продвигаясь мелкими шажками, строго по одной изолированной команде с мгновенным скриншот-контролем, мы успешно реализовали стартовый блок нового руководства на чистой tom_1:

- Шаг 0.1. Запустили полное обновление системы (sudo pacman -Syu –noconfirm).
- Шаг 0.2. Отправили ВМ в чистый ребут (sudo reboot), зашли обратно и зафиксировали в памяти ОЗУ эталонное родное ядро 7.0.9-arch2-1.
- Шаг 0.3. Проверили пакет физического микрокода (pacman -Q linux-firmware) — версия 20260519-1 на месте, реальные сетевухи Intel/Broadcom заведутся.
- Шаг 0.4. Ювелирно создали и сохранили на диск универсальный сетевой конфиг /etc/systemd/network/20-wired.network с маской Name=en* eth* и статикой 192.168.1.150/24.
- Шаг 0.5. Перевели сетевую службу в автозапуск: sudo systemctl enable systemd-networkd (симлинки созданы).
- Шаг 0.6. Включили DNS-резолвер: sudo systemctl enable systemd-resolved.
- Шаг 0.7. Убедились, что удаленный доступ активен: systemctl is-enabled sshd рапортует enabled.
- Шаг 0.8. Проверили автозапуск веб-сервера: systemctl is-enabled nginx выдает enabled.
- Шаг 0.9. Прогнали синтаксический тест Nginx (sudo nginx -t) — успешно, порт 5000 жестко привязан в nginx.conf.
- Шаг 0.10. Проверили хэши безопасности пользователей (sudo getent shadow root eva) — современный Yescrypt (y) на месте, учетки активны. Прямой вход под root через su - root проверен и защищен вашим личным паролем.
- Шаг 0.11. Вывели параметры ZRAM на хосте (lsmod и zramctl) — модуль в ядре активен, swap на 4 ГБ функционирует штатно.
- Шаг 0.12. Сделали резервную копию рабочей таблицы разделов: sudo cp /etc/fstab /etc/fstab.bak [1.9].
- Шаг 0.13. Обнулили оригинал: sudo truncate -s 0 /etc/fstab [1.9]. Контрольный cat /etc/fstab вернул идеальную пустую строку (оригинальная Btrfs-структура субтомов хоста надежно заперта в бэкапе .bak весом 837 байт) [1.4, 1.9].
- Шаг 0.14. Отработали нашу новую эдит-запись: одной чистой командой доставили инструменты сжатия и генератор ZRAM (sudo pacman -S –noconfirm squashfs-tools zram-

generator) [1.4]. Раздельная проверка подтвердила их наличие в системе [1.4].
- Шаг 0.15. Доставили программную основу нашего будущего веб-инсталлятора: sudo pacman -S –noconfirm php-fpm [1.4].
- Шаг 0.16–0.18. Поймали «выключенный» статус PHP-FPM через is-enabled и принудительно перевели его в автозапуск (sudo systemctl enable php-fpm), добившись жесткого статуса enabled [1.4].

ЧАСТЬ 4. ЧТО ПРЕДСТОИТ СДЕЛАТЬ (ВЕБ-ИНСТАЛЛЯТОР)

Мы остановились ровно перед разделом «Создание структуры каталогов для конструктора ISO». Мы полностью утвердили новую изящную концепцию: наша флешка будет не просто Live-CD, а полноценным WebUI-автономным сервером установки. Окно SSH нам нужно только для контроля, а сама установка на tom_2 будет происходить по клику кнопки из браузера на веб-странице 192.168.1.150:5000!

Предстоящие шаги доработки руководства на tom_1:

1. Создание структуры каталогов конструктора:

Мы создадим папку под слепок системы mkdir -p ~/custom_iso/arch/x86_64/. Проверим систему tom_2 и официальный iso образ на необходимость создания директори изолированного пути для ядра загрузчика, которого ранее в доноре возможно не было:
и если его не было,то создадим

#bash

```
mkdir -p ~/custom_iso/arch/boot/x86_64/
```

и проверим создание.

2. Разработка Веб-Фронтенда и Бэкенда (HTML / CSS / JS / PHP):

В каталоге /usr/share/nginx/html/ мы заменим стандартную заглушку Nginx на пульт управления:

1. index.html + style.css: красивый интерфейс с большой кнопкой «Начать установку системы на tom_2».
2. script.js: обработчик клика, который через асинхронный fetch() пингует бэкенд и выводит логи установки в реальном времени.
3. install.php: скрипт, принимающий запрос от JS и инициирующий системный вызов через shell_exec('sudo /usr/share/nginx/html/installer.sh sda 2>&1');.

3. Обход мины с правами доступа (http пользователь):

Поскольку PHP-FPM в Arch работает от имени пользователя http, мы создадим изолированный файл прав /etc/sudoers.d/web-installer внутри конструктора ISO и пропишем туда строго одну строчку беспарольного доступа к скрипту:

text

```
http ALL=(ALL:ALL) NOPASSWD: /usr/share/nginx/html/installer.sh
```

http ALL=(ALL:ALL) NOPASSWD: /usr/share/nginx/html/installer.sh

4. Создание скрипта разметки по логике LABEL (Полный отказ от UUID):

Мы напишем installer.sh, который примет диск sda, занулит его через sgdisk –zap-all и наречет разделы жесткими глобальными метками: EFI раздел → LABEL=«ARCH_BOOT» (FAT32), Root раздел → LABEL=«ARCH_OS» (Btrfs). Скрипт создаст ваши btrfs-субтома (/@, /@home, /@pkg, /@log), скопирует систему из ОЗУ и подкинет универсальный статический /etc/fstab, полностью завязанный на метки LABEL=.

5. Запуск заморозки системы:

Выполним команду mksquashfs / ~/custom_iso/arch/x86_64/airootfs.sfs .... Внутри этого слепка теперь гарантированно окажутся и модули ядра 7.0.9, и php-fpm, и zram-generator. Сразу после этого вернем fstab хоста на место.

6. Сборка через xorriso:

Мы соберем новый 102-й образ, жестко привязав имя диска к вашей метке тома -volid «ARCH_202605», пропишем флаг ядра unknown_nmi_panic=0 для защиты от прерываний Hyper-V

Модификация параметров загрузчика под Hyper-V

- 6.1. Добавление флага ядра: При формировании конфигурационного файла ~/custom_iso/loader/entries/01-archiso-linux.conf в строку параметров ядра (options) дописать:

text

```
textunknown_nmi_panic=0
```

Это заблокирует ложные аппаратные прерывания виртуализации Hyper-V и предотвратит ступор консоли.

Этап 3. Развертывание и изоляция веб-бэкенда (Nginx + PHP)

- 3.1. Интеграция PHP: Установить пакет php-fpm внутри слепка системы и связать его с Nginx через сокет в nginx.conf.
- 3.2. Настройка прав веб-пользователя: Создать изолированный файл правил прав доступа по пути /etc/sudoers.d/web-installer внутри конструктора ISO.
- 3.3. Беспарольный доступ к Bash: Прописать в этот файл строго одну строчку:

text

```
texthttp ALL=(ALL:ALL) NOPASSWD: /usr/share/nginx/html/installer.sh
```

- 3.4. Маскировка прав: Перед заморозкой SquashFS жестко выставить права на этот файл, иначе sudo его проигнорирует:

#bash

```
bashchown root:root /etc/sudoers.d/web-installer
chmod 0440 /etc/sudoers.d/web-installer
```

Этап 4. Разработка WebUI-инсталлятора и логики LABEL

- 4.1. Внедрение Bash-исполнителя: Написать скрипт /usr/share/nginx/html/installer.sh, который:
  - Принимает имя целевого диска как аргумент (например, $1 → sda).
  - Стирает старую разметку через sgdisk –zap-all.
  - Размечает диск и присваивает разделам жесткие метки: Раздел 1 (FAT32) → ARCH_BOOT, Раздел 2 (Btrfs) → ARCH_OS.
  - Создает внутри ARCH_OS копию вашей структуры субтомов (/@, /@home, /@pkg, /@log).Копирует файлы из текущего ОЗУ флешки на новые субтома без интернета.
- 4.2. Полный отказ от UUID: Внедрить в устанавливаемую систему универсальный статический /etc/fstab, полностью завязанный на метки:

text

```
LABEL=ARCH_OS   /       btrfs   rw,noatime,compress=zstd,subvol=/@
0
LABEL=ARCH_BOOT /boot   vfat    rw,relatime,fmask=0022,dmask=0022
2
```

- 4.3. Верстка интерфейса (Фронтенд): Заменить дефолтный index.html в папке Nginx на кастомный пульт управления с использованием HTML, CSS и JavaScript.
- 4.4. Кнопка «Старт»: Написать install.php, который по клику JS-кнопки из браузера безопасно вызывает:

install.php

```
phpshell_exec('sudo /usr/share/nginx/html/installer.sh sda 2>&1');
```

Конец

Дальше читаем только если 23.05.26 вы собрали флешку, но она не запустилась!

1. Глобальная цель проекта

Собрать кастомный, полностью автономный ISO-образ Arch Linux на базе живой системы tom_1 для офлайн-установки на «слепые» физические сервера (Supermicro, старый HP, два кастомных ноунейма) без интернета и мониторов. Доступ к серверам после старта с флешки — строго по SSH.

Если 23.05.26 года до 23:59 вам не хватит ума собрать рабочую флешку. Только тогда! Для Анализа читаем рабочий код от 21.05.26

Остановились тут

Создаем точки монтирования, монтируем ISO и копируем его структуру в наш рабочий каталог.

#bash

```
mkdir -p /tmp/iso_mount
mkdir -p ~/custom_iso
sudo mount -o loop ~/archlinux-x86_64.iso /tmp/iso_mount
cp -r /tmp/iso_mount/EFI ~/custom_iso/
cp -r /tmp/iso_mount/loader ~/custom_iso/
cp -r /tmp/iso_mount/arch ~/custom_iso/
```

Команда для проверки результата:

Убедимся, что папки теперь на месте в ~/custom_iso:

#bash

```
ls -la ~/custom_iso
```

Монтирование прошло успешно (предупреждение о read-only для ISO — это норма). Теперь копируем структуру во временную папку.

Выполните копирование:

#bash

```
cp -r /tmp/iso_mount/EFI ~/custom_iso/
cp -r /tmp/iso_mount/loader ~/custom_iso/
cp -r /tmp/iso_mount/arch ~/custom_iso/
```

Даем права на запись во всю структуру конструктора

Сделаем все папки и файлы внутри ~/custom_iso доступными для изменения:

#bash

```
chmod -R +w ~/custom_iso
```

Чтобы лично убедиться, что права изменились и у пользователя eva теперь есть доступ на запись (w) ко всем папкам и файлам внутри конструктора, выполните две команды проверки:

#bash

```
chmod -R +w ~/custom_iso
ls -la ~/custom_iso
ls -la ~/custom_iso/loader/entries/
```

На что смотрим в выводе перед тем, как идти дальше:

- В строках папок arch, EFI и loader права должны начинаться с drwxr-xr-x (главное — наличие буквы w в первой тройке символов).
- Внутри папки entries файлы должны получить права -rw-r–r– (также с буквой w)

Записываем универсальный конфиг в правильный файл

#bash

```
cat << 'EOF' > ~/custom_iso/loader/entries/01-archiso-linux.conf
title Arch Linux install medium (x86_64, UEFI)
linux /arch/boot/x86_64/vmlinuz-linux
initrd /arch/boot/x86_64/initramfs-linux.img
options archisobasedir=arch archisolabel=ARCH_202605
EOF
```

Проверяем результат записи

Убедимся, что файл 01-archiso-linux.conf обновился корректно:

#bash

```
cat ~/custom_iso/loader/entries/01-archiso-linux.conf
```

Проверка пройдена идеально.

Конфигурационный файл 01-archiso-linux.conf успешно перезаписан. Параметр archisolabel=ARCH_202605 зафиксирован в системе. Теперь ядро при старте будет искать носитель строго по метке, и загрузка не упадет.

Делаем fstab чистым (через бэкап)

временно уберем строчки из fstab на tom_1, соберем слепок и вернем всё назад. На чистом tom_1 в fstab и так практически ничего нет, кроме корня.

Проверяем содержимое оригинального fstab

#bash

```
cat /etc/fstab
```

Делаем резервную копию вашего рабочего fstab:

#bash

```
sudo cp /etc/fstab /etc/fstab.bak
```

проверяем его наличие

#bash

```
ls -la /etc/fstab /etc/fstab.bak
cat /etc/fstab.bak
```

Мы должны увидеть два файла одинакового размера, и содержимое fstab.bak должно до символа совпадать с оригиналом.

Бэкап проверен

всё на месте до единого символа. В оригинальном конфиге как раз жестко завязаны UUID разделов Btrfs (/, /home, /var/...) и vfat (/boot) диска sdb. Именно эти строки systemd отчаянно пытался найти в виртуалке, вызывая аварийный режим. Теперь, когда в основном файле пусто, система внутри ISO-образа больше не будет спотыкаться о локальную разметку tom_1. Оригинальные данные в полной безопасности, переходим к упаковке.

Полностью очищаем fstab перед упаковкой:

#bash

```
sudo truncate -s 0 /etc/fstab
```

Базовая загрузочная структура полностью готова к упаковке новой системы. Переходим к установке утилит и заморозке чистого tom_1 на место удаленного файла ОС.

Установка squashfs-tools

Поскольку система чистая, ставим пакет сжатия заново:

#bash

```
sudo pacman -S --noconfirm squashfs-tools
```

Замораживаем и упаковываем живой tom_1

После успешной установки пакета запускаем сборку слепка файловой системы:

#bash

```
sudo mksquashfs / ~/custom_iso/arch/x86_64/airootfs.sfs \
-e /proc /sys /dev /run /tmp /mnt /media /lost+found ~/archlinux-
x86_64.iso ~/custom_iso \
-comp zstd -b 1M
```

(Процесс займет несколько минут, на экране побегут упаковываемые файлы).

Команда для обязательной проверки результата:

Как только утилита mksquashfs завершит работу и вернет терминал к строке [eva@tom1 ~]$, проверьте, что файл создался и посмотрите его вес:

#bash

```
ls -lh ~/custom_iso/arch/x86_64/airootfs.sfs
```

Упаковка SquashFS завершена успешно.

Файл airootfs.sfs весит 1.5 ГБ, статистика UID/GID на скриншоте подтверждает, что все пользователи (включая eva и http) корректно перенеслись внутрь слепка системы.
Поскольку утилита mksquashfs работала через sudo, владельцем созданного файла стал root. Проверяем и правим права, чтобы на этапе финальной сборки ISO у нас не возникло ошибок доступа.

возвращаем ваш fstab на место, чтобы tom_1 не пострадал:

#bash

```
sudo mv /etc/fstab.bak /etc/fstab
```

Меняем владельца файла

Выполните команду смены владельца:

#bash

```
sudo chown eva:eva ~/custom_iso/arch/x86_64/airootfs.sfs
```

Обязательная проверка результата

Убедимся, что права обновились корректно:

#bash

```
ls -lh ~/custom_iso/arch/x86_64/airootfs.sfs
```

На что смотрим в выводе:

Вместо root root в строке файла должно быть четко написано eva eva.

Проверка прав пройдена успешно.

Владельцем файла airootfs.sfs теперь официально является eva eva.
Мы подошли к финальному этапу — сборке кастомного ISO-образа. Помня прошлую ошибку с регистром букв в имени загрузчика (BOOTx64.EFI), мы используем точную и проверенную команду сборки.
Поскольку система чистая, сначала установим xorriso, а затем соберем образ.

Установка xorriso

Выполните команду установки:

#bash

```
sudo pacman -S --noconfirm xorriso
```

Финальная сборка универсального ISO

Запустите команду сборки (в ней прописана правильная маленькая буква x и наша метка тома ARCH_202605):

#bash

```
xorriso -as mkisofs \
```

```
  -iso-level 3 \
  -full-iso9660-filenames \
  -volid "ARCH_202605" \
  -eltorito-alt-boot \
  -e "EFI/BOOT/BOOTx64.EFI" \
  -no-emul-boot \
  -isohybrid-gpt-basdat \
  -output ~/arch_custom.iso \
  ~/custom_iso
```

Обязательная проверка результата

Убедимся, что файл arch_custom.iso успешно сгенерировался в вашей домашней директории, проверив его размер:

#bash

```
ls -lh ~/arch_custom.iso
```

Сборка нового образа завершена идеально.

Мы видим заветную строку Writing to 'stdio...' completed successfully и готовый файл arch_custom.iso весом 1.7 ГБ.
В этот раз мы проверили каждый шаг, исправили права на папки и зашили универсальный параметр загрузки ядра по метке тома archisolabel=ARCH_202605.
Переходим к проверке на виртуальной машине tom_2. Вытаскиваем готовый ISO-образ в Windows.

Скачиваем новый ISO в Windows

Эта команда без лишнего мусора и длинных списков выведет ровно одно значение — текущий IP-адрес, по которому tom_1 доступен в вашей сети.

#bash

```
ip route get 1 | awk '{print $7}'
```

Откройте PowerShell на вашем хостовом компьютере с Windows и запустите скачивание файла:

cmd

```
scp eva@192.168.1.72:/home/eva/arch_custom.iso
C:\Users\Public\Downloads\arch_custom.iso
```

(При запросе введите пароль пользователя eva).

выполнен успешно!

Новый универсальный образ arch_custom.iso размером 1.7 ГБ полностью скачан в Windows. Дата изменения — свежая (14:01).

Открываем папку с файлом на Windows

Нажмите на клавиатуре комбинацию клавиш Win + R (откроется окошко «Выполнить»), вставьте туда вот этот путь и нажмите Enter:

txt

```
C:\Users\Public\Downloads
```

дважды щелкнуть мышкой по диску

и записать руфусом

Извлекаем диск из Windows

Даже если вы нажимали «Извлечь», Rufus или Проводник могли оставить фоновый процесс.

1. Откройте «Управление дисками» (Disk Management) в Windows.
2. Найдите внизу списка наш Диск 2 (виртуальный).
3. Нажмите правой кнопкой мыши по серой зоне с надписью «Диск 2» и выберите «Отсоединить виртуальный жесткий диск». Если его там уже нет — отлично.