

схемы

Сквозная схема файловой системы хоста tom_1 и конструктора ISO

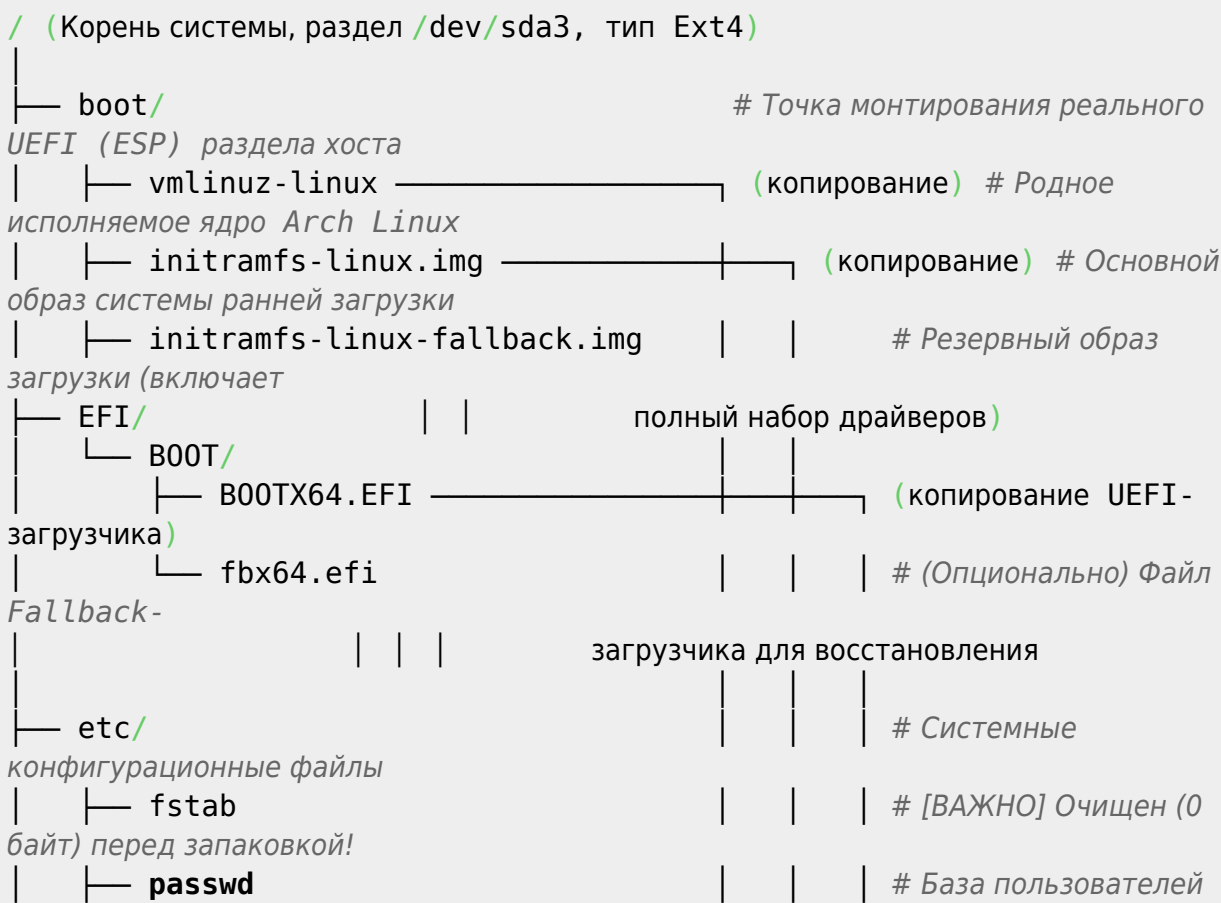
- Конструктор как изоляция: Каталог ~/custom_iso/ создается пользователем eva локально на хосте. Он служит сборочным цехом. Всё, что находится внутри ~/custom_iso/, станет корнем файловой системы самой флешки после записи ISO.
- Процесс клонирования ядра: Команды cp /boot/... ~/custom_iso/... берут текущие рабочие загрузочные файлы хоста tom_1 и переносят их во вложенную структуру конструктора, чтобы будущий образ гарантированно загрузился на том же ядре.
- Рекурсивное исключение при запаковке: Когда утилита mksquashfs сканирует корень / для генерации файла airootfs.sfs, она упаковывает все файлы etc, usr/share/nginx/html и т.д., но в параметрах запуска обязательно пропускает папку ~/custom_iso (и архивы *.iso), чтобы избежать бесконечного цикла (упаковки архива внутрь самого себя).

Директория ~/custom_iso/ не входит внутрь файла airootfs.sfs и не подключается к нему напрямую. Она выполняет роль внешнего «контейнера» (конструктора), внутри которого этот файл создается и хранится.

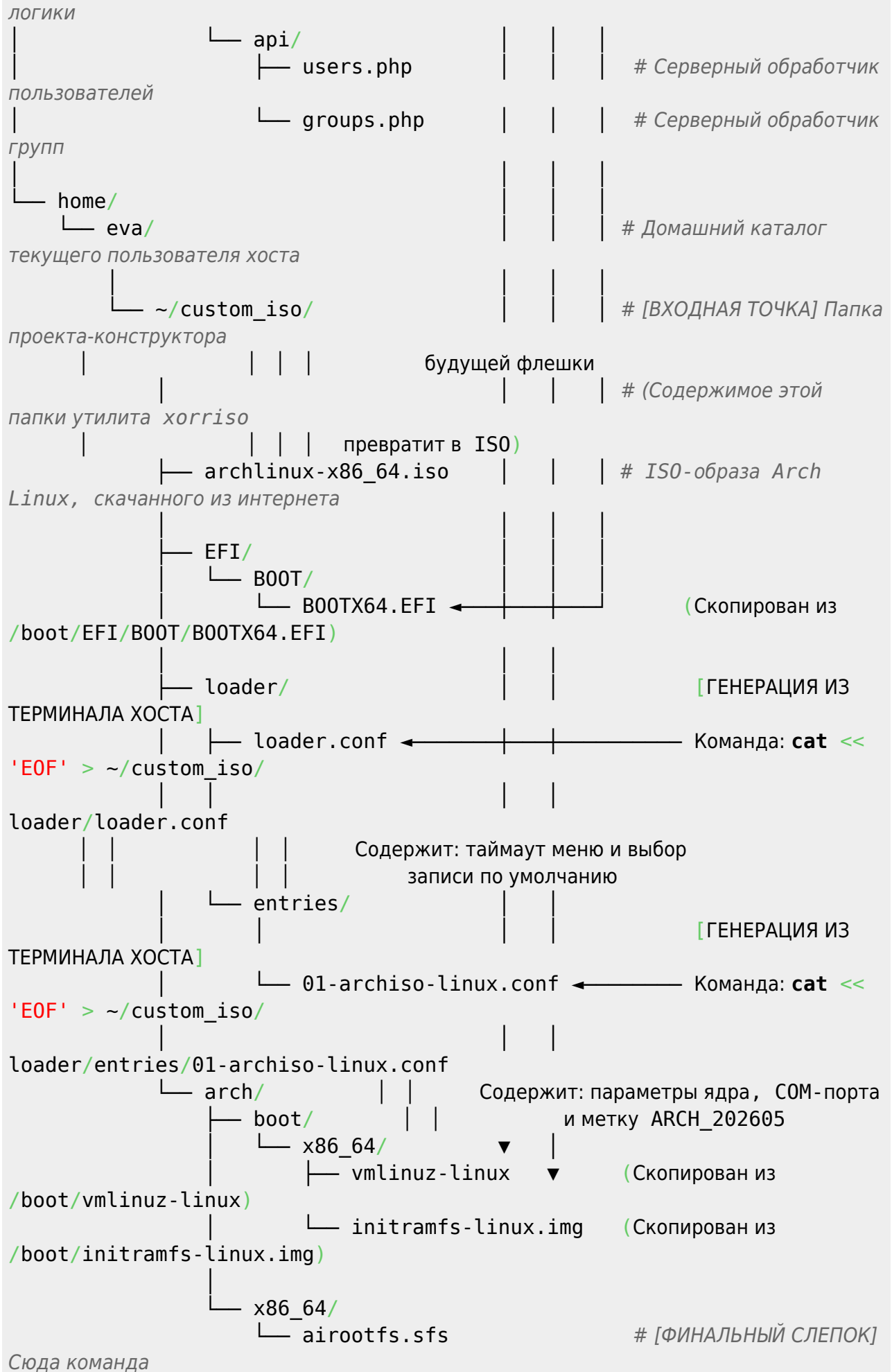
Если сопоставить обе схемы, то папка ~/custom_iso/ находится на самом верхнем уровне и в момент финальной сборки через утилиту xorriso превращается в корень загрузочного ISO-образа.

Вот схема, показывающая точное местоположение airootfs.sfs внутри конструктора ~/custom_iso/:

circuit



(используется <code>api/users.php</code>)			
— group			# База групп
(используется <code>api/groups.php</code>)			
— shadow			# Хэши паролей
(модифицируется веб-интерфейсом)			
— sudoers			# Настроен беспарольный
доступ пользователя <code>http</code>			
— nginx/			
— nginx.conf			# Веб-сервер Nginx
настроен на порт 5000			
— systemd/			
— network/			
— 20-wired.network			# Сетевой конфиг
(статический IP 192.168.1.150)			
— system/			
			# Включена автозагрузка
служб: <code>nginx</code> , <code>php-fpm</code> ,			
			systemd-
<code>networkd</code> , <code>systemd-resolved</code> , <code>smb</code>			
— samba/			
— smb.conf			# Файловый сервер для
Windows-разработки			
— var/cache/pacman/pkg/			# Кэш менеджера
пакетов			
— [*].pkg.tar.zst			# Загруженные из
интернета пакеты (<code>nginx</code> ,			
			php-fpm,
<code>samba...</code>)			
— usr			
— bin/			# Системные
исполняемые файлы			
— pacman			# Менеджер пакетов
хоста (утилита управления)			
— share/			
— nginx/			
— html/			# Исходный код Веб-
Инсталлятора (права 775/664,			
— index.html			http:http)
			# Интерфейс панели
управления			
— css/			
— style.css			# Стили оформления
панели			
— js/			
— app.js			# Клиентские скрипты



```

# mksquashfs запакует
весь корень '/',
# исключая виртуальные
ФС и сам custom_iso
# (Опционально) Файл
Fallback-загрузчика
|
| для восстановления
|
| # При загрузке сервера ядро монтирует этот файл в
Live-ОЗУ как корень '/':
|— etc/ (passwd, shadow, nginx.conf, 20-
wired.network)
|— usr/share/nginx/html/ (код вашего веб-
инсталлятора)
└ ...
```

Архитектурная логика связей на схеме:

- Конструктор как изоляция: Каталог `~/custom_iso/` создается пользователем `eva` локально на хосте. Он служит сборочным цехом. Всё, что находится внутри `~/custom_iso/`, станет корнем файловой системы самой флешки после записи ISO.
- Процесс клонирования ядра: Команды `cp /boot/... ~/custom_iso/...` берут текущие рабочие загрузочные файлы хоста `tom_1` и переносят их во вложенную структуру конструктора, чтобы будущий образ гарантированно загрузился на том же ядре.
- Рекурсивное исключение при запаковке: Когда утилита `mksquashfs` сканирует корень `/` для генерации файла `airootfs.sfs`, она упаковывает все файлы `etc`, `usr/share/nginx/html` и т.д., но в параметрах запуска обязательно пропускает папку `~/custom_iso` (и архивы `*.iso`), чтобы избежать бесконечного цикла (упаковки архива внутрь самого себя).
- Стрелки на схеме в точности повторяют логику выполнения следующих трех команд
 - `cp /boot/EFI/BOOT/BOOTX64.EFI ~/custom_iso/EFI/BOOT/BOOTX64.EFI`
 - `cp /boot/vmlinuz-linux ~/custom_iso/arch/boot/x86_64/vmlinuz-linux`
 - `cp /boot/initramfs-linux.img ~/custom_iso/arch/boot/x86_64/initramfs-linux.img`
- файлы конфигурации загрузчика `loader.conf` и `01-archiso-linux.conf` не копируются из готовых папок хоста `tom_1`. Они создаются с нуля на лету с помощью консольных команд перенаправления вывода (`cat « 'EOF' > ...`) прямо внутри целевых папок конструктора `~/custom_iso/`.
- Менеджер пакетов `pacman (/usr/bin/pacman)`: Он является частью установленной операционной системы хоста `tom_1`. При запуске команды `mksquashfs` бинарный файл `pacman` и все его конфигурационные файлы автоматически попадают внутрь слепка `airootfs.sfs`. Благодаря этому на целевом сервере `tom_2` в Live-режиме будет полноценно работать команда `pacman`.
- Загруженные пакеты (`/var/cache/pacman/pkg/`): Когда на Шаге 2.2 выполняется команда `sudo pacman -S nginx php-fpm ...`, пакеты сначала скачиваются из интернета в этот системный кэш, а затем устанавливаются в систему. Если перед сборкой образов не вызвать очистку кэша (`pacman -Sc`), все эти скачанные `.tar.zst` файлы тоже будут запакованы внутрь `airootfs.sfs`, что увеличит размер финального ISO-образа.

- Скачанный оригинальный ISO-образ (~/archlinux-x86_64.iso): В данном руководстве автор реализует метод сборки на основе слепка живой системы (Host Snapshot), а не классический метод через утилиту archiso. Поэтому оригинальный ISO-образ Arch Linux автора не интересует — из него ничего не монтируется и не копируется. В тексте руководства он упоминается ровно один раз на Шаге 4.2 в качестве флага исключения: ... -e /proc /sys ... ~/archlinux-x86_64.iso ~/custom_iso ... Это сделано исключительно для того, чтобы архиватор mksquashfs случайно не стал упаковывать этот огромный скачанный файл весом в 1 ГБ внутрь создаваемого слепка.

Исключенные директории (Флаг -e в mksquashfs)

При сборке слепка командой `sudo mksquashfs / ...` напрямую исключаются виртуальные, временные и резервные папки хоста, так как они генерируются ядром автоматически при загрузке в ОЗУ:

- /proc и /sys — виртуальные ФС ядра.
- /dev — файлы устройств (заполняются службой udev).
- /run — динамические runtime-данные (сюда при старте создается сокет /run/php-fpm/php-fpm.sock).
- /tmp — временные файлы.
- /mnt и /media — точки монтирования внешних дисков.

circuit

```

/boot/                                     # Точка монтирования реального
UEFI (ESP) раздела хоста
├─ vmlinuz-linux                           # Родное исполняемое ядро Arch
Linux (сжатый бинарник)
├─ initramfs-linux.img                     # Основной образ файловой системы
ранней загрузки (ОЗУ-диск)
├─ initramfs-linux-fallback.img            # Резервный образ загрузки
(включает полный набор драйверов)
└─ EFI/
    └─ BOOT/
        ├─ BOOTX64.EFI                    # Основной бинарный файл
загрузчика UEFI (в данном проекте systemd-boot)
        └─ fbx64.efi                      # (Опционально) Файл Fallback-
загрузчика для восстановления

```

Ключевые файлы схемы:

- vmlinuz-linux и initramfs-linux.img — это «сердце» загрузки. Они обновляются каждый раз, когда вы выполняете команду `sudo pacman -Syu`. Именно их актуальные версии забирает конструктор, чтобы гарантировать отсутствие паники ядра на флешке.
- EFI/BOOT/BOOTX64.EFI — это переименованный исполняемый файл systemd-boot, который материнская плата сервера считывает первым при старте ПК в режиме UEFI

разбивки диска оригинальго Arch linux с Fat32

На основе стандартных рекомендаций по установке оригинального Arch Linux в режиме UEFI (включая требования из руководства по созданию ISO), ниже представлена схема разбивки физического диска (например, /dev/sda или /dev/nvme0n1).
Схема содержит обязательный системный раздел FAT32 (для загрузчика), основной раздел Ext4 (корень системы) и раздел подкачки Swap:

Физический диск (Разметка таблицы разделов: GPT)

GPT

Номер раздела	Файловая система	Название / Назначение	Точка монтирования
/dev/sda1	FAT32	EFI System Partition (ESP)	/boot
	Размер: 512 МБ - 1 ГБ		
/dev/sda2	linux-swap	Swap Partition (Подкачка)	[SWAP]
	Размер: равен ОЗУ или выше		
/dev/sda3	Ext4	Root Partition (Корень)	/(Сюда ставится система)
	Размер: Всё оставшееся		

Важные технические особенности:

- Почему FAT32 обязателен? Материнские платы с UEFI «из коробки» не умеют читать файловые системы Linux (такие как Ext4, Btrfs или SquashFS). Они понимают только FAT32. Поэтому папка /boot выносится на отдельный FAT32-раздел, чтобы плата могла запустить загрузчик BOOTX64.EFI и считать ядро.
- Флаг загрузки: При создании раздела /dev/sda1 утилитами fdisk или gdisk ему обязательно выставляется тип EFI System (код ef00 в gdisk), а форматируется он строго командой mkfs.vfat -F 32 /dev/sda1.
- Использование в ISO: При сборке вашего кастомного ISO-образа утилита xorriso имитирует эту структуру прямо внутри одного ISO-файла, создавая скрытый FAT-образ для совместимости с UEFI-загрузкой целевых серверов.

1.9.1 Примеры схем

UEFI с GPT

Точка монтирования в установленной системе	Раздел	Тип раздела	Рекомендуемый размер
/boot1	/dev/системный_раздел_efi	Системный раздел EFI	1 ГиБ
[SWAP]	/dev/раздел_подкачки	Linux swap	Не менее 4 ГиБ
/	/dev/корневой_раздел	Linux x86-64 root (/)	Остаток, минимум 23-32 ГиБ

Схема веб-панели управления в основной системе (tom_1)

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)

From:

<https://wwoss.ru/> - worldwide open-source software

Permanent link:

https://wwoss.ru/doku.php?id=software:linux_server:arch_linux:arch_linux_circuit

Last update:

2026/05/25 17:35

