Mkinitcpio (Русский)

Эта страница нуждается в сопроводителе

Статья не гарантирует актуальность информации. Помогите русскоязычному сообществу поддержкой подобных страниц. См. **Команда переводчиков ArchWiki**

Ссылки по теме

- **systemd (Русский)**
- **Kernel modules (Русский)**

Contents

[hide]

Введение

mkinitcpio это Bash скрипт используемый для создания начального загрузочного диска системы. Из **mkinitcpio man page**:

> *The initial ramdisk is in essence a very small environment (early userspace) which loads various kernel modules and sets up necessary things before handing over control to init. This makes it possible to have, for example, encrypted root file systems and root file systems on a software RAID array. mkinitcpio allows for easy extension with custom hooks, has autodetection at runtime, and many other features.*

Традиционно ядро отвечает за обнаружение всего оборудования и выполняет задачи на ранних этапах **процесса загрузки** до монтирования корневой файловой системы.

В настоящее время корневая файловая система может быть на широком диапазоне аппаратных средств от SCSI до SATA и USB дисков, управляемых различными контроллерами от разных производителей. Кроме того корневая файловая система может быть зашифрована или сжата, находиться в RAID массиве или группе логических томов. Простой способ справиться с этой сложностью является передача управления в пользовательском пространстве: начальный загрузочный диск.

Смотрите также: **/dev/brain0 » Blog Archive » Early Userspace in Arch Linux**.

mkinitcpio является модульным инструментом для построения initramfs CPIO образа, предлагая много преимуществ по сравнению с альтернативными методами. Эти преимущества включают в себя:

- Использование **BusyBox**, чтобы обеспечить легкость базы для раннего userspace'а.
- Поддержка **udev** для автоматического обнаружения оборудования, предотвращая загрузку ненужных модулей.
- Использование расширяемого hook-based скрипта с поддержкой пользовательских хуков, которые могут быть включены в состав пакетов и устанавливаться с помощью **pacman**.
- Поддержка **LVM2**, **dm-crypt** как и LUKS томов, **mdadm**, и **swsusp**, и **suspend2** для использования спящего режима и загрузки с USB носителей.
- Предоставляет много возможностей для настройки из командной строки ядра без необходимости пересборки образа.

mkinitcpio создан разработчиками Arch Linux и вкладами сообщества. Смотрите **public Git repository**.

Установка mkinitcpio

Установить пакет **`mkinitcpio`**, который является зависимым от пакета **`linux`**, поэтому большинство пользователей уже установили его.

Продвинутые пользователи могут захотеть установить последнюю версию mkinitcpio из Git с пакетом **`mkinitcpio-git`**[AUR].

Note: Настоятельно читать **arch-projects mailing list** при использовании mkinitcpio из Git!

Создание загрузочного образа

По умолчанию mkinitcpio генерирует два образа после установки или обновления ядра: `/boot/initramfs-linux.img` and `/boot/initramfs-linux-fallback.img`. *fallback* образ создается с точно таким же конфигурационным файлом за исключением хука **autodetect**, что позволяет включить в него все модули.
Хук **autodetect** обнаруживает нужные модули необходимые для оборудования и включает их в initramfs.

Можно создать любое количество initramfs с различными конфигурациями. Необходимый initramfs должен быть прописан в **конфигурационном файле загрузчика**[broken link: invalid section]. После изменения конфигурационного файла initramfs должен быть пересобран. Для стандартного ядра Arch Linux, **`linux`**:

```
# mkinitcpio -p linux
```

Параметр `-p` (сокращение от *preset*) указывает на использование preset файла из `/etc/mkinitcpio.d` (т.е. `/etc/mkinitcpio.d/linux.preset` для `linux`). preset файл определяет параметры сборки initramfs образа вместо указания файла конфигурации и выходной файл каждый раз.

Warning: *preset* файлы используются для автоматической пересборки initramfs после обновления ядра. Будьте внимательны при их редактировании.

Варианты создания initcpio

Пользователи могут вручную создать образ с помощью альтернативного конфигурационного файла. Например, следующее будет генерировать initramfs образ в соответствии с `/etc/mkinitcpio-custom.conf` и сохранит его в `/boot/linux-custom.img`.

```
# mkinitcpio -c /etc/mkinitcpio-custom.conf -g /boot/linux-custom.img
```

Если необходимо создать образ с ядром отличным от загруженного.Доступные версии ядер можно посмотреть в `/usr/lib/modules`.

```
# mkinitcpio -g /boot/linux.img -k 3.3.0-ARCH
```

Настройка

`/etc/mkinitcpio.conf` - основной конфигурационный файл **mkinitcpio**. Кроме того, в каталоге `/etc/mkinitcpio.d` располагаются preset файлы (e.g. `/etc/mkinitcpio.d/linux.preset`).

Warning: lvm2, **mdadm**, и **encrypt НЕ ВКЛЮЧЕНЫ** по умолчанию. Прочитайте эту страницу чтобы узнать как их включить и настроить.

Note: Если у вас более одного контроллера дисков использующих одно пространство имен устройств (например 2 SCSI/SATA или IDE контроллера) и они требуют разных модулей, укажите правильный порядок в `MODULES` в файле `/etc/mkinitcpio.conf`, иначе устройства могут поменяться местами и вы получите kernel panic при загрузке. Более правильный путь - использовать **persistent block device naming**, чтобы быть уверенным, что вы смонтировали действительно то, что хотели.

Note: PS/2 keyboard users: Для того, чтобы получить ввод с клавиатуры в начале загрузки, если он не работает, необходимо добавить **keyboard** хук в `HOOKS`. На некоторых материнских платах (в основном древних, но и некоторых новых), контроллер i8042 не может быть определен автоматически. Вы можете обнаружить эту ситуацию заранее. Если у вас есть порт PS/2 и появляется сообщение `i8042: PNP: No PS/2 controller found. Probing ports directly`, добавьте **atkbd** в `MODULES`.

Можно изменять шесть переменных в конфигурационном файле:

`MODULES`
: Модули ядра, которые будут загружены до выполнения хуков.

`BINARIES`
: Дополнительные исполняемые файлы, которые необходимо включить в initramfs.

`FILES`
: Дополнительные файлы, которые необходимо включить в initramfs.

`HOOKS`

Hooks - скрипты, выполняемые в initramfs.

`COMPRESSION`

Тип используемого сжатия initramfs.

`COMPRESSION_OPTIONS`

Дополнительные опции архиваторов. Использование этого параметра не рекомендуется. mkinitcpio будет обрабатывать особые требования к архиваторам (например `--check=crc32` для xz), что может привести к невозможности запуска системы.

MODULES

Указывает какие модули ядра должны быть загружены прежде чем что-либо будет сделано.

Для модулей с суффиксом `?` не будет выводиться ошибка, если они не будут найдены. Это может быть полезно для пользовательских ядер.

BINARIES and FILES

Указывают какие файлы необходимо добавить в initramfs. `BINARIES` и `FILES` будут добавлены до запуска хуков и использоваться для переопределения файлов использаемых хуками. `BINARIES` - бинарные файлы из `PATH`, необходимые для работы библиотеки будут автоматически добавлены. `FILES` добавляет файлы как есть. Например:

```
FILES="/etc/modprobe.d/modprobe.conf"
BINARIES="kexec"
```

И в `BINARIES`, и в `FILES` может быть добавлено несколько файлов с пробелом в качестве разделителя.

HOOKS

Параметр `HOOKS` наиболее важный в файле настроек. Хуки - это небольшие скрипты, которые описывают что будет добавлено к образу, а также дополнительные действия, выполняемые при загрузке системы. Хуки указываются по имени и выполняются по порядку.

Значение по умолчанию для `HOOKS` должно быть достаточным для большинства простых установок с одним диском. Для корневых устройств, которые являются многоуровневыми или многоблочными устройствами, такими как **LVM**, **mdadm** или **dm-crypt**, см. соответствующие страницы вики для дальнейшей необходимой конфигурации.

Build hooks

Build hooks - хуки сборки. Располагаются в `/usr/lib/initcpio/install`. Эти файлы используются mkinitcpio во время сборки initramfs. Должны содержать две функции: `build`и `help`. Функция `build` перечисляет модули, файлы, исполняемые файлы, которые добавляются в образ. Функция `help` содержит описание действий хука.

Для получения списка всех доступных хуков:

```
$ mkinitcpio -L
```

Используйте опцию `-H` для вывода информации о конкретном хуке:

```
$ mkinitcpio -H udev
```

Runtime hooks

Runtime hooks - хуки периода выполнения располагаются в `/usr/lib/initcpio/hooks`, пользовательские хуки могут быть помещены в `/etc/initcpio/hooks` Для любого хука периода исполнения всегда должен быть хук сборки с тем же именем, в котором имеется

вызов `add_runscript`, указывающий на добавление хука периода исполнения в образ. Эти файлы обрабатываются командным интерпретатором ash из busybox во время раннего пользовательского пространства. Они запускаются по порядку записи в `HOOKS` за исключением хуков очистки. Runtime хуки могут содержать несколько функций:

`run_earlyhook`: Функции с таким именем будут запускаться однажды после установки API файловых систем и обработки командной строки ядра. Как правило, здесь запускаются дополнительные демоны, такие как udev, необходимые для раннего процесса загрузки.

`run_hook`: Функции с таким именем запускаются вскоре после ранних хуков. Это наиболее распространенная точка хуков, и здесь должны выполняться операции, такие как сборка многоуровневых блочных устройств.

`run_latehook`: Функции с этим именем запускаются после установки корневого устройства. Это следует использовать, умеренно, для дальнейшей настройки корневого устройства или для установки других файловых систем, таких как `/usr`.

`run_cleanuphook`: Функции с этим именем запускаются как можно позже, и в порядке обратном, порядку их перечисления в `HOOKS` файла конфигурации. Эти хуки должны использоваться для любой очистки в последнюю минуту, например, для закрытия всех демонов, запущенных ранними хуками.

Доступные хуки

Таблица стандартных хуков и как они влияют на создание и выполнение образа. Обратите внимание, что эта таблица не является полной, так как пакеты могут предоставлять свои хуки.

Current hooks			
busybox	**systemd**	**Установка**	**Запуск**
base		Устанавливает все начальные каталоги, базовые утилиты и библиотеки. Всегда ставьте этот хук первым, за исключением случаев, когда вы действительно знаете, что делаете. Предоставляет busybox recovery shell при использовании совместно с хуком **systemd**.	--
udev	**systemd**	Добавляет udev в образ ram-диска	Udev автоматически создает файл устройства для корня и загружает необходимые модули для его работы. Рекомендуется использовать.
usr		Добавляет поддержку отделного `/usr` раздела.	Монтирует раздел `/usr` после монтирования корневой файловой системы.
resume		--	Необходим для работы спящего режима (suspend to disk). Работает с *swsusp*.

			Хук **systemd** поддерживает только *swsusp*. См. **Hibernation**.
btrfs	--	Sets the required modules to enable **Btrfs** for using multiple devices with Btrfs. This hook is not required for using Btrfs on a single device.	Runs `btrfs device scan` to assemble a multi-device Btrfs root file system when **udev** hook or **systemd** hook is not present. The **btrfs-progs** package is required for this hook.
autodetect		меньшает размер initramfs пытаясь определить какие модули вам нужны. Проверьте список модулей которые он добавил. Он должен запускаться раньше других подсистем. Все обработчики выполняемые до него будут включать все модули.	--
modconf		Добавляет конфигурационные файлы modprobe `/etc/modprobe.d` и `/usr/lib/modprobe.d`	--
block		Добавляет все модули блочных устройств, ранее предоставляемые другими хуками (*fw*, *mmc*, *pata*, *sata*, *scsi*, *usb*, *virtio*).	--
pcmcia		Добавляет pcmcia модули. Требует **pcmciautils**[ссылка недействительна: package not found].	--
net	*не реализован*	Добавляет поддержку сети. Для PCMCIA устройств добавьте хук **pcmcia**.	Требуется для корневой файловой системы через NFS.
dmraid	*?*	Поддержка корневой файловой системы на fakeRAID массивах. Необходимо установить пакет **dmraid**. Обратите внимание, что необходимо использовать `mdadm` с хуком **mdadm_udev** с fakeRAID если ваш контроллер поддерживает это.	Находит и монтирует fakeRAID блочные устройства используя `dmraid`.
mdadm	--	Обеспечивает поддержку для сборки RAID массивов из `/etc/mdadm.conf`, или автоопределеним во время загрузки. Необходимо установить пакет **mdadm**. Вместо этого хука предпочтительнее использовать хук **mdadm_udev**.	Находит и собирает программные RAID блочные устройства с помощью `mdassemble`.
mdadm_ude		Обеспечивает поддержку для	Находит и собирает

v		сборки RAID массивов с помощью udev. Необходимо установить пакет **mdadm**. Если вы используете этот крючок с массивом FakeRAID, рекомендуется включить `mdmon` в секции BINARIES и добавить хук **shutdown** чтобы избежать восстановления массива при перезагрузке.	программные RAID блочные устройства с помощь `udev` и `mdadm`.Это предпочтительный метод использования mdadm (заменяет хук *mdadm*).
keyboard		Добавляет модули необходимые для работы клавиатур. Используйте этот хук, если необходимо использовать USB клавиатуру на ранней стадии загрузки (в initramfs).	--
keymap	**sd-vconsole**	Добавляет в initramfs раскладки указанные `/etc/vconsole.conf`.	Загружает раскладки указанные в `/etc/vconsole.conf`.
consolefont		Добавляет в initramfs консольный шрифт указанный `/etc/vconsole.conf`.	Загружает шрифт указанный в `/etc/vconsole.conf`.
encrypt	**sd-encrypt**	Добавлет модуль ядра `dm_crypt` и `cryptsetup`. Требуется пакет **cryptsetup**.	Определяет и подключает зашифрованый корневой раздел. См. **#Runtime customization**[broken link: invalid section] для дальнейшей настройки. Для **sd-encrypt** см. **systemd-cryptsetup-generator(8)** допустимые параметры ядра. В качестве альтернативы, если файл `/etc/crypttab.initramfs` существует, он будет добавлен в initramfs как `/ etc / crypttab`. Дополнительную информацию о синтаксисе crypttab см. **crypttab(5)**.
lvm2	**sd-lvm2**	Добавляет поддержку `lvm`. Требует установленного пакета **lvm2**.	Включает поддержку lvm. Необходим, если корневая файловая система на **LVM**.
fsck		Добавляет исполняемый файл fsck и необходимые обработчики файловых систем. Если стоит после	Запускает fsck для корневой файловой системы (и раздела `/usr`) до монтирования.

	хука **autodetect**, то будут добавлены только обработчики для вашей корневой файловой системы. Использование этого хука настоятельно рекомендуется и **обязательно** с отдельным `/usr`разделом.	Использование этого хука требует, чтобы rw передавался в параметрах ядра (**discussion**).
filesystems	Включает в образ модули необходимых файловых систем. Этот хук **обязателен**, если Вы не указываете модули файловых систем в MODULES.	--

COMPRESSION

Ядро поддерживает несколько форматов для сжатия initramfs - **`gzip`**, **`bzip2`**, lzma, **`xz`** (также известный как lzma2), **`lzo`** , И **`lz4`**. Для большинства случаев использования gzip, lzop и lz4 обеспечивают наилучший баланс сжатого размера изображения и скорости декомпрессии. Стандартный `mkinitcpio.conf` имеет различные опции `COMPRESSION`. Раскомментируйте один, чтобы выбрать необходимый формат сжатия.

Отсутствие параметра `COMPRESSION` приведет к файлу initramfs с сжатием gzip. Чтобы создать несжатый образ, укажите `COMPRESSION = cat` в конфигурации или используйте `-z cat` в командной строке.

Убедитесь, что для метода, который вы хотите использовать, установлена правильная утилита сжатия файлов.

COMPRESSION_OPTIONS

Это дополнительные флаги, переданные программе, указанной `COMPRESSION`, например:

```
COMPRESSION_OPTIONS = '- 9'
```

В общем, они никогда не понадобятся, поскольку mkinitcpio будет следить за тем, чтобы любой поддерживаемый метод сжатия имел необходимые флаги для создания рабочего изображения. Кроме того, неправильное использование этого параметра может привести к не загружаемой системе, если ядро не сможет распаковать результирующий архив.

Изменения строки параметров ядра

Не зависимо от наличия initramfs некоторые опции приходится передавать через строку параметров ядра, как например корневое устройство. Некоторые из обработчиков mkinitcpio имеют специальные опции. Они и обсуждаются в этой главе.

Если вы не знаете, что такое строка параметров ядра, читайте документацию по **GRUB** или **LILO**.

Вход в безотказный режим

Если Вы добавите опцию

```
break=y
```

в строку параметров ядра, то init остановится после инициализации и вы получите *dash* шелл. Это может быть использовано, чтобы проверить, что все хорошо. После выхода загрузка продолжится в обычном режиме.

Запрет обработчиков

Вы можете запретить обработчики при помощи параметра *disablehooks* в строке параметров ядра:

```
disablehooks=hook1,hook2,hook2
```

например,

```
disablehooks=resume
```

Запрет загрузки модулей

Вы можете запретить загрузку некоторых модулей добавив параметр *disablemodules* в строку параметров ядра:

```
disablemodules=mod1,mod2,mod3
```

например,

```
disablemodules=ata_piix
```

Использование raid

Добавьте обработчик raid в список HOOKS в /etc/mkinitcpio.conf

Параметры ядра: Укажите md массивы с помощью: md= parameter: (см. ниже). Простого добавления вашего raid массива достаточно.

```
 Пример: md=0,/dev/sda3,/dev/sda4 md=1,/dev/hda1,/dev/hdb1
```

Затем добавьте следующее в строку kernel в **grub/menu.lst**:

```
 Пример: md=0,/dev/sda3,/dev/sda4 md=1,/dev/hda1,/dev/hdb1
```

Т.е. это будет выглядеть так:

```
 kernel /vmlinuz26beyond root=/dev/md0 ro md=0,/dev/sda1,/dev/sdb1
```

Эта строка создает два md массива с постоянными суперблоками

Настройка:

```
  - для старых raid массивов без постоянных суперблоков:
     md=<номер устройства md>,<уровень raid>,<chunk size factor>,<fault
level>,dev0,dev1
  - для raid массивом с постоянными суперблоками:
     md=<номер устройства md>,dev0,dev1,...,devn
  - для сборки partitionable массивов:
     md=d<md device no.>,dev0,dev1,...,devn
```

Параметры:

```
- <номер устройства md> = номер устройства:
   0 значит md0, 1 значит md1, ...
- <уровень raid> = -1 линейный режим, 0 striped режим
   другие режимы поддерживаются только с постоянным суперблоком
- <chunk size factor> = (только для raid-0 и raid-1):
   Установить chunk size as 4k << n.
- <fault level> = игнорируется
- <dev0-devn>: т.е. /dev/hda1,/dev/hdc1,/dev/sda1,/dev/sdb1
```

Использование net

Параметры ядра:

ip=

Описание интерфейса может быть в короткой форме, которая состоит просто из имени интерфейса (eth0 или какой либо еще), или в длинной форме. Длинная форма содержит семь элементов, разделенных двоеточием:

```
ip=<client-ip>:<server-ip>:<gw-ip>:<netmask>:<hostname>:<device>:<autoconf>
nfsaddrs= это алиас для ip= и может использоваться также.
```

Разъяснение параметров:

```
  <client-ip>   IP адрес клиента. Если пустой, то автоматически назначается
                протоколом RARP/BOOTP/DHCP. Какой протокол используется
                зависит от параметра <autoconf>. Если его значение не пустое,
                то autocnf работает корректно.

  <server-ip>   IP адрес NFS сервера. Если используется RARP для определения
                адреса клиента и этот параметр не пуст, то принимаются ответы
                только от указанного сервера.
                Для использования разных RARP и NFS серверов,
                укажите ваш RARP сервер здесь (или оставьте пустым), и
                укажите ваш NFS сервер в параметре `nfsroot'
                (см. выше). Если тут пусто, будет использован сервер,
ответивший
                на RARP/BOOTP/DHCP запрос.

  <gw-ip>       IP адрес шлюза если сервер в другой подсети. Если тут пусто,
                предполагается, что сервер находится в одной подсети, если
значение
                не получено по BOOTP/DHCP.

  <netmask>     Маска подсети для локального интерфейса. Если тут пусто,
                используется маска по умолчанию в зависимости от класса IP,
```

```
                если конечно не переопределена значением из BOOTP/DHCP
ответа.

  <hostname>    Имя машины клиента. Если пусто, используется IP адрес в виде
ASCII
                строки, или значение полученное по BOOTP/DHCP.

  <device>      Имя сетевого устройства. Если тут пусто, все устройства будут
использованы
                для RARP/BOOTP/DHCP запросов, и все настройки будут получены
из первого
                пришедшего ответа. Если вы имеете только одно сетевое
устройство, можно спокойно
                оставить поле пустым.

  <autoconf>   Какой метод использовать для автонастройки: 'rarp', 'bootp',
или 'dhcp'.
                Если значение 'both', 'all' или пусто, все протоколы будут
использованы.
                'off', 'static' или 'none' означают запрет автонастройки.
```

Примеры:

```
  ip=127.0.0.1:::::lo:none  --> Разрешить lo интерфейс.
  ip=192.168.1.1:::::eth2:none --> Статический eth2 интерфейс.
  ip=:::::eth0:dhcp --> Разрешить dhcp протокол для настройки интерфейса
eth0.
```

nfsroot=

Если параметр 'nfsroot' НЕ передан, будет использовано значение по умолчанию "/tftpboot/%s".

```
  nfsroot=[<server-ip>:]<root-dir>[,<nfs-options>]
```

Описание параметров:

```
  <server-ip>   IP адрес NFS сервера. Если параметр отсутствует
                адрес по умолчанию определяется по переменной
                `ip' (см. ниже). Одно из применений этого параметра -
                использование разных серверов для
                RARP и NFS. Обычно можете оставить этот параметр пустым.

  <root-dir>    Путь к директории, которая будет корневой на клиенте. Если
тут
```

```
                написано "%s", то оно будет заменено на ASCII представление
IP
                клиента.

  <nfs-options> Стандартные опции NFS. Все опции разделены запятыми.
                Если опция отсутствует будут использованы следующие значения
по умолчанию:
                        port            = как скажет portmap демон на сервере
                        rsize           = 1024
                        wsize           = 1024
                        timeo           = 7
                        retrans         = 3
                        acregmin        = 3
                        acregmax        = 60
                        acdirmin        = 30
                        acdirmax        = 60
                        flags           = hard, nointr, noposix, cto, ac
```

root=/dev/nfs

```
  Если вы не используете параметр nfsroot=, вам нужно установить
root=/dev/nfs
  для загрузки с автонастройкой корня в nfs.
```

Использование lvm

Если ваш корень находится на lvm, вы должны добавить обработчик **lvm2**. Также, вы должны передать имя корневого устройства ядру в формате

```
root=/dev/mapper/<имя-группы-томов>-<логическое-имя-тома>
```

например

```
root=/dev/mapper/myvg-root
```

Использование шифрования на корневом разделе

Если ваш корневой раздел зашифрован, вы должны добавить обработчик **encrypt**. Затем укажите ядру корневой раздел также, как если бы он не был зашифрован.

Например для sata/scsi:

```
root=/dev/sda5
```

или для зашифрованного lvm:

```
root=/dev/mapper/myvg-root
```

Корневое устройство автоматически поменяется на */dev/mapper/root*.

Использование LUKS томов

Если вы используете LUKS для шифрования дисков, скрипт инициализации поймет это автоматически, если вы указали обработчик **encrypt**. Он спросит пароль для разблокирования тома.

Если этого не происходит, попробуйте добавить filesystem-module в список модулей в фале /etc/mkinitcpio.conf если он не вкомпилирован в ядро.

Использование legacy cryptsetup томов

Если вы используете legacy cryptsetup том, вы должны указать все опции, необходимые для его разблокировки в строке параметров ядра. Опции формата

```
crypto=hash:cipher:keysize:offset:skip
```

представляют опции cryptsetup: --hash, --cipher, --keysize, --offset и --skip. Если вы пропустите опцию, будет использовано значение по умолчанию, т.е. вы можете написать просто

```
crypto=::::
```

если вы создали том с настройками по умолчанию.

ЗАМЕЧАНИЕ: По техническим причинам невозможно проверить корректность пароля для legacy cryptsetup тома. Если вы ошибетесь, у вас просто не получится его смонтировать. Рекомендуется использовать LUKS вместо legacy cryptsetup.

Использование loop-aes томов

mkinitcpio пока не поддерживает loop-aes.

Troubleshooting

Extracting the image

If you are curious about what is inside the initrd image, you can extract it and poke at the files inside of it.

The initrd image is an SVR4 CPIO archive, generated via the `find` and `bsdcpio` commands, optionally compressed with a compression scheme understood by the kernel. For more information on the compression schemes, see **#COMPRESSION**.

mkinitcpio includes a utility called `lsinitcpio` which will list and extract the contents of initramfs images.

You can list the files in the image with:

```
$ lsinitcpio /boot/initramfs-linux.img
```

And to extract them all in the current directory:

```
$ lsinitcpio -x /boot/initramfs-linux.img
```

You can also get a more human-friendly listing of the important parts in the image:

```
$ lsinitcpio -a /boot/initramfs-linux.img
```

Recompressing a modified extracted image

After extracting an image as explained above, after modifying it, you can find the command necessary to recompress it. Edit `/usr/bin/mkinitcpio` and change the line as shown below (line 531 in mkinitcpio v20-1.)

```
#MKINITCPIO_PROCESS_PRESET=1 "$0" "${preset_cmd[@]}"
MKINITCPIO_PROCESS_PRESET=1 /usr/bin/bash -x "$0" "${preset_cmd[@]}"
```

Then running `mkinitcpio` with its usual options (typically `mkinitcpio -p linux`), toward the last 20 lines or so you will see something like:

```
+ find -mindepth 1 -printf '%P\0'
+ LANG=C
+ bsdcpio -0 -o -H newc --quiet
+ gzip
```

Which corresponds to the command you need to run, which may be:

```
# find -mindepth 1 -printf '%P\0' | LANG=C bsdcpio -0 -o -H newc --quiet |
gzip > /boot/initramfs-linux.img
```

Warning: It is a good idea to rename the automatically generated /boot/initramfs-linux.img before you overwrite it, so you can easily undo your changes. Be prepared for making a mistake that prevents your system from booting. If this happens, you will need to boot through the fallback, or a boot CD, to restore your original, run mkinitcpio to overwrite your changes, or fix them yourself and recompress the image.

"/dev must be mounted" when it already is

The test used by mkinitcpio to determine if /dev is mounted is to see if /dev/fd/ is there. If everything else looks fine, it can be "created" manually by:

```
# ln -s /proc/self/fd /dev/
```

(Obviously, /proc must be mounted as well. mkinitcpio requires that anyway, and that is the next thing it will check.)

Using systemd HOOKS in a LUKS/LVM/resume setup

Using `systemd`/`sd-encrypt`/`sd-lvm2` **HOOKS** instead of the traditional `encrypt`/`lvm2`/`resume` requires different initrd parameters to be passed by your **boot loader**. See **this post on forum** for details.

Possibly missing firmware for module XXXX

When initramfs are being rebuild after a kernel update, you might get these two warnings:

```
==> WARNING: Possibly missing firmware for module: aic94xx
==> WARNING: Possibly missing firmware for module: smsmdtv
```

These appear to any Arch Linux users, especially those who have not installed these firmware modules. If you do not use hardware which uses these firmwares you can safely ignore this message.

mkinitcpio creates images with all the shared libraries missing

If your machine fails to boot with an "Attempted to kill init!" kernel panic right off the bat (before any `init` or `systemd`-related messages appear on the screen), and running `lsinitcpio` reveals that all the shared libraries are missing from the images generated in `/boot`, make sure there is a symbolic link at `/usr/lib64` pointing to `/usr/lib`, and rebuild them all.

Standard rescue procedures

With an improper initial ram-disk a system often is unbootable. So follow a system rescue procedure like below:

Загрузка выполняется на одной машине и терпит неудачу на другой

`autodetect` hook скрипта *mkinitcpio* фильтрует ненужные **kernel modules** в первичном initramfs путем сканирования /sys и модулей, загруженных во время запуска. Если вы переносите /boot каталог на другую машину и последовательность загрузки терпит неудачу на стадии early userspace, то это может происходить, потому что новое аппаратное обеспечение не определено отсутствующими модулями ядра. Обратите внимание, что для USB 2.0 и 3.0 нужны разные модули ядра.

Чтобы исправить, сначала попробуйтей выбрать fallback образ в вашем **загрузчике**, поскольку он не фильтруется с помощью `autodetect`. После загрузки выполните *mkinitcpio* на новой машине, чтобы пересобрать первичный образ с корректными модулями. Если fallback образ не решил проблему, пропробуйте загрузиться в Arch Linux live CD/USB, выполнить chroot в установленную систему и выполнить *mkinitcpio* на новой машине. В крайнем случае, попробуйте **вручную** добавить модули в initramfs.

Я параноик!

Если вам действительно нужно знать, что включено в initrd, вы можете вытаскивать и класть внутрь образа файлы. Это просто cpio архив, но...

Предупреждение о cpio

Использование cpio для извлечения опасно (от root'а), так как оно пытается извлечь файлы в /, ломая существующую систему загрузки. Для исправления, вам придется переставить все пакеты, файлы которых были перезаписаны с помощью pacman.static. Список файлов выдаст команда bsdtar -t -f kernel26.img

Использование bsdtar для извлечения

bsdtar извлекает файлы в текущий каталог.

Используйте команду:

```
bsdtar -x -f /boot/kernel26.img
```

Categories:

- Boot process (Русский)
- Kernel (Русский)
- Arch projects (Русский)
- Commands (Русский)
- Русский