mkinitcpio

mkinitcpio is the next generation of **initramfs** creation.

Related articles

systemd

Kernel modules

Minimal initramfs

Boot debugging

Contents

Overview

mkinitcpio is a Bash script used to create an initial ramdisk environment. From the **`mkinitcpio(8)` (https://jlk.fjfi.cvut.cz/arch/manpages/man/mkinitcpio.8)**:

> 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.

Traditionally, the kernel was responsible for all hardware detection and initialization tasks early in the **boot process** before mounting the root file system and passing control to `init`. However, as technology advances, these tasks have become increasingly complex.

Nowadays, the root file system may be on a wide range of hardware, from SCSI to SATA to USB drives, controlled by a variety of drive controllers from different manufacturers. Additionally, the root file system may be encrypted or compressed; within a software RAID array or a logical volume group. The simple way to handle that complexity is to pass management into userspace: an initial ramdisk.

See also: **/dev/brain0 » Blog Archive » Early Userspace in Arch Linux (https://web.archive.org/web/20150430223035/http://archlinux.me/brain0/2010/02/13/early-userspace-in-arch-linux/)**.

mkinitcpio is a modular tool for building an initramfs CPIO image, offering many advantages over alternative methods; these advantages include:

- The use of **BusyBox (http://www.busybox.net/)** to provide a small and lightweight base for early userspace.

- Support for **udev** for hardware auto-detection at runtime, thus preventing the loading of unnecessary modules.
- Using an extendable hook-based init script, which supports custom hooks that can easily be included in packages.
- Support for **LVM2**, **dm-crypt** for both plain and LUKS volumes, **RAID**, and **swsusp** for resuming, and booting from USB mass storage devices.
- The ability to allow many features to be configured from the kernel command line without needing to rebuild the image.

mkinitcpio has been developed by the Arch Linux developers and from community contributions. See the **public Git repository (https://projects.archlinux.org/mkinitcpio.git/)**.

Installation

Install the `mkinitcpio (https://www.archlinux.org/packages/?name=mkinitcpio)` package, which is a dependency of the `linux (https://www.archlinux.org/packages/?name=linux)` package, so most users will already have it installed.

Advanced users may wish to install the latest development version of mkinitcpio from Git with the `mkinitcpio-git (https://aur.archlinux.org/packages/mkinitcpio-git/)`AUR package.

Note: It is **highly** recommended that you follow the **arch-projects mailing list (https://mailman.archlinux.org/mailman/listinfo/arch-projects)** if you use mkinitcpio from Git!

Image creation and activation

By default, the mkinitcpio script generates two images after kernel installation or upgrades: a *default* image, and a *fallback* image that skips the *autodetect* hook thus including a full range of mostly-unneeded modules. This is accomplished via the *preset* files which most kernel packages install in `/etc/mkinitcpio.d/` (e.g. `/etc/mkinitcpio.d/linux.preset` for `linux`). A preset is a predefined definition of how to create an initramfs image instead of specifying the configuration file and output file every time. The `-p` / `--preset` switch specifies a *preset* to utilize. For example, `mkinitcpio -p linux` selects the preset provided by the **`linux (https://www.archlinux.org/packages/?name=linux)`** package.

An additional configuration file is located at `/etc/mkinitcpio.conf` and is used to specify options global to all presets. The `-P` / `--allpresets` switch specifies that all presets should be utilized when regenerating the initramfs after a `mkinitcpio.conf` change.

Users may create any number of initramfs images with a variety of different configurations. The desired image must be specified in the respective **boot loader** configuration file.

Warning: *preset* files are used to automatically regenerate the initramfs after a kernel update; be careful when editing them.

Generate customized manual initcpio

Users can generate an image using an alternative configuration file. For example, the following will generate an initramfs image according to the directions in `/etc/mkinitcpio-custom.conf` and save it at `/boot/linux-custom.img`.

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

If generating an image for a kernel other than the one currently running, add the kernel version to the command line. You can see available kernel versions in `/usr/lib/modules/`.

```
# mkinitcpio -g /boot/linux-custom2.img -k 3.3.0-ARCH
```

Configuration

The primary configuration file for **mkinitcpio** is `/etc/mkinitcpio.conf`. Additionally, preset definitions are provided by kernel packages in the `/etc/mkinitcpio.d` directory (e.g. `/etc/mkinitcpio.d/linux.preset`).

Warning: lvm2, **mdadm**, and **encrypt** are **NOT** enabled by default. Please read this section carefully for instructions if these hooks are required.

Note:

- Users with multiple hardware disk controllers that use the same node names but different kernel modules (e.g. two SCSI/SATA or two IDE controllers) should use **persistent block device naming** to ensure that the right devices are mounted. Otherwise, the root device location may change between boots, resulting in kernel panics.
- **PS/2 keyboard users**: In order to get keyboard input during early init, if you do not have it already, add the **keyboard** hook to the `HOOKS`. On some motherboards (mostly ancient ones, but also a few new ones), the i8042 controller cannot be automatically detected. It is rare, but some people will surely be without keyboard. You can detect this situation in advance. If you have a PS/2 port and get `i8042: PNP: No PS/2 controller found. Probing ports directly` message, add **atkbd** to the `MODULES`.

Users can modify six variables within the configuration file:

`MODULES`
: Kernel modules to be loaded before any boot hooks are run.

`BINARIES`
: Additional binaries to be included in the initramfs image.

`FILES`
: Additional files to be included in the initramfs image.

`HOOKS`
: Hooks are scripts that execute in the initial ramdisk.

`COMPRESSION`

Used to compress the initramfs image.

`COMPRESSION_OPTIONS`

Extra arguments to pass to the `COMPRESSION` program. Usage of this setting is strongly discouraged. mkinitcpio will handle special requirements for compressors (e.g. passing `--check=crc32` to xz), and misusage can easily lead to an unbootable system.

MODULES

The MODULES array is used to specify modules to load before anything else is done.

Modules suffixed with a `?` will not throw errors if they are not found. This might be useful for custom kernels that compile in modules which are listed explicitly in a hook or config file.

Note:

- If using **reiser4**, it *must* be added to the modules list.
- If you will be needing any file system during the boot process that is not live when you run mkinitcpio—for example, if your LUKS encryption key file is on an **ext2** file system but no **ext2** file systems are mounted when you run mkinitcpio—that file system module must also be added to the MODULES list. See **Dm-crypt/System configuration#cryptkey** for more details.

BINARIES and FILES

These options allow users to add files to the image. Both `BINARIES` and `FILES` are added before hooks are run, and may be used to override files used or provided by a hook. `BINARIES` are auto-located within a standard `PATH` and are dependency-parsed, meaning any required libraries will also be added. `FILES` are added *as-is*. For example:

```
FILES=(/etc/modprobe.d/modprobe.conf)
```

```
BINARIES=(kexec)
```

Note that for both `BINARIES` and `FILES`, multiple entries can be added delimited with spaces.

HOOKS

The `HOOKS` setting is the most important setting in the file. Hooks are small scripts which describe what will be added to the image. For some hooks, they will also contain a runtime component which provides additional behavior, such as starting a daemon, or assembling a stacked block device. Hooks are referred to by their name, and executed in the order they exist in the `HOOKS` setting in the config file.

The default `HOOKS` setting should be sufficient for most simple, single disk setups. For root devices which are stacked or multi-block devices such as **LVM**, **mdadm**, or **dm-crypt**, see the respective wiki pages for further necessary configuration.

Build hooks

Build hooks are found in `/usr/lib/initcpio/install/`, custom build hooks can be placed in `/etc/initcpio/install/`. These files are sourced by the bash shell during runtime of mkinitcpio and should contain two functions: `build` and `help`. The `build` function describes the modules, files, and binaries which will be added to the image. An API, documented by mkinitcpio(8) (https://jlk.fjfi.cvut.cz/arch/manpages/man/mkinitcpio.8), serves to facilitate the addition of these items. The `help` function outputs a description of what the hook accomplishes.

For a list of all available hooks:

```
$ mkinitcpio -L
```

Use mkinitcpio's `-H`/`--hookhelp` option to output help for a specific hook, for example:

```
$ mkinitcpio -H udev
```

Runtime hooks

Runtime hooks are found in `/usr/lib/initcpio/hooks/`, custom runtime hooks can be placed in `/etc/initcpio/hooks/`. For any runtime hook, there should always be a build hook of the same name, which calls `add_runscript` to add the runtime hook to the image.

These files are sourced by the busybox ash shell during early userspace. With the exception of cleanup hooks, they will always be run in the order listed in the `HOOKS` setting. Runtime hooks may contain several functions:

`run_earlyhook`: Functions of this name will be run once the API file systems have been mounted and the kernel command line has been parsed. This is generally where additional daemons, such as udev, which are needed for the early boot process are started from.

`run_hook`: Functions of this name are run shortly after the early hooks. This is the most common hook point, and operations such as assembly of stacked block devices should take place here.

`run_latehook`: Functions of this name are run after the root device has been mounted. This should be used, sparingly, for further setup of the root device, or for mounting other file systems, such as `/usr`.

`run_cleanuphook`: Functions of this name are run as late as possible, and in the reverse order of how they are listed in the `HOOKS` setting in the config file. These hooks should be used for any last minute cleanup, such as shutting down any daemons started by an early hook.

Common hooks

A table of common hooks and how they affect image creation and runtime follows. Note that this table is not complete, as packages can provide custom hooks.

busybox init	**systemd init**	**Build hook**	**Runtime hook (busybox init only)**
base		Sets up all initial directories and installs base utilities and libraries. Always keep this hook as the first hook unless you know what you are doing, as it provides critical busybox init when not using **systemd** hook. Provides a busybox recovery shell when using **systemd** hook.	--
udev	**systemd**	Adds udevd, udevadm, and a small subset of udev rules to your image.	Starts the udev daemon and processes uevents from the kernel; creating device nodes. As it simplifies the boot process by not requiring the user to explicitly specify necessary modules, using it is recommended.
usr		Adds support for `/usr` on a separate partition.	Mounts the `/usr` partition after the real root has been mounted.
resume		--	Tries to resume from the "suspend to disk" state. See **Hibernation** for further configuration.
btrfs	--	Sets the required modules to enable **Btrfs** for using multiple devices with Btrfs. You need to have **btrfs-progs (https://www.archlinux.org/packages/?name=btrfs-progs)** installed to use this. 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 (https://www.archlinux.org/packages/?name=btrfs-progs)** package is required for this hook.
autodetect		Shrinks your initramfs to a smaller size by creating a whitelist of modules from a scan of sysfs. Be sure to verify included modules are correct and none are missing. This hook must be run before other subsystem hooks in order to take advantage of auto-detection. Any hooks placed before 'autodetect' will be installed in full.	--
modconf		Includes modprobe configuration files from `/etc/modprobe.d/` and `/usr/lib/modprobe.d/` .	--
block		Adds all block device modules, formerly separately provided by the *fw*, *mmc*, *pata*, *sata*, *scsi*, *usb*, and *virtio* hooks.	--
pcmcia		Adds the necessary modules for PCMCIA devices. You need to have **pcmciautils (https://www.archlinux.org/packages/?name=pcmciautils)** installed to use this.	--
net	*not implemented*	Adds the necessary modules for a network device. You must have **mkinitcpio-nfs-utils (https://www.archlinux.org/packages/?name=mkinitcpio-nfs-utils)** installed to use this. For PCMCIA net devices, please add the **pcmcia** hook too.	Provides handling for an NFS-based root file system.
dmraid	*?*	Provides support for fakeRAID root devices. You must have **dmraid (https://www.archlinux.org/packages/?name=dmraid)** installed to use this. Note that it is preferred to use `mdadm` with the **mdadm_udev** hook with fakeRAID if your controller supports it.	Locates and assembles fakeRAID block devices using `dmraid` .
mdadm	--	Provides support for assembling RAID arrays from `/etc/mdadm.conf` , or autodetection during boot. You must have **mdadm (https://www.archlinux.org/p**	Locates and assembles software RAID block devices using `mdassemble` .

		ackages/?name=mdadm) installed to use this. The **mdadm_udev** hook is preferred over this hook.	
mdadm_udev		Provides support for assembling RAID arrays via udev. You must have `mdadm (https://www.archlinux.org/packages/?name=mdadm)` installed to use this. If you use this hook with a FakeRAID array, it is recommended to include `mdmon` in the binaries section.	Locates and assembles software RAID block devices using `udev` and `mdadm` incremental assembly. This is the preferred method of mdadm assembly (rather than using the above *mdadm* hook).
keyboard		Adds the necessary modules for keyboard devices. Use this if you have an USB or serial keyboard and need it in early userspace (either for entering encryption passphrases or for use in an interactive shell). As a side effect, modules for some non-keyboard input devices might be added too, but this should not be relied on. Supersedes old *usbinput* hook. **Tip:** For systems that are booted with different hardware configurations (e.g. laptops with external keyboard vs. internal keyboard or **headless systems**), it is helpful to place this hook before **autodetect** in order to always include all keyboard drivers. Otherwise the external keyboard only works in early userspace if it was connected when creating the image.	--
keymap	**sd-vconsole**	Adds the specified keymap(s) from `/etc/vconsole.conf` to the initramfs. If you use system encryption, especially **full disk encryption**, make sure you add it before the `encrypt` hook.	Loads the specified keymap(s) from `/etc/vconsole.conf` during early userspace.
consolefont		Adds the specified console font from `/etc/vconsole.conf` to the initramfs.	Loads the specified console font from `/etc/vconsole.conf` during early userspace.
encrypt	**sd-encrypt**	Adds the `dm_crypt` kernel module and the `cryptsetup` tool to the image. You must have `cryptsetup (https://www.archlinux.org/packages/?name=cryptsetup)` installed to use this.	Detects and unlocks an encrypted root partition. See **#Runtime customization** for further configuration. For **sd-encrypt** see **dm-crypt/System configuration#Using sd-encrypt hook**.
lvm2	**sd-lvm2**	Adds the device mapper kernel module and the `lvm` tool to the image. You must have `lvm2 (https://www.archlinux.org/packages/?name=lvm2)` installed to use this.	Enables all LVM2 volume groups. This is necessary if you have your root file system on **LVM**.
fsck		Adds the fsck binary and file system-specific helpers. If added after the **autodetect** hook, only the helper specific to your root file system will be added. Usage of this hook is **strongly** recommended, and it is required with a separate `/usr` partition.	Runs fsck against your root device (and `/usr` if separate) prior to mounting. The use of this hook requires the rw parameter to be set on the kernel commandline (**discussion (https://bbs.archlinux.org/viewtopic.php?pid=1307895#p1307895)**).
filesystems		This includes necessary file system modules into your image. This hook is **required** unless you specify your file system modules in MODULES.	--

COMPRESSION

The kernel supports several formats for compression of the initramfs—`gzip (https://www.archlinux.org/packages/?name=gzip)`, `bzip2 (https://www.archlinux.org/packages/?name=bzip2)`, lzma, `xz (https://www.archlinux.org/packages/?name=xz)` (also known as lzma2), `lzo (https://www.archlinux.org/packages/?name=lzo)`, and `lz4 (https://www.archlinux.org/packages/?name=lz4)`. For most use cases, gzip, lzop, and lz4 provide the best balance of compressed image size and decompression speed. The provided `mkinitcpio.conf` has the various `COMPRESSION` options commented out. Uncomment one to choose which compression format you desire.

Specifying no `COMPRESSION` will result in a gzip-compressed initramfs file. To create an uncompressed image, specify `COMPRESSION=cat` in the config or use `-z cat` on the command line.

Make sure you have the correct file compression utility installed for the method you wish to use.

COMPRESSION_OPTIONS

These are additional flags passed to the program specified by `COMPRESSION`, such as:

```
COMPRESSION_OPTIONS=(-9)
```

In general these should never be needed as mkinitcpio will make sure that any supported compression method has the necessary flags to produce a working image. Furthermore, misusage of this option can lead to an unbootable system if the kernel is unable to unpack the resultant archive.

Runtime customization

Runtime configuration options can be passed to `init` and certain hooks via the kernel command line. Kernel command-line parameters are often supplied by the bootloader. The options discussed below can be appended to the kernel command line to alter default behavior. See **Kernel parameters** and **Arch boot process** for more information.

init from base hook

`root`

This is the most important parameter specified on the kernel command line, as it determines what device will be mounted as your proper root device. mkinitcpio is flexible enough to allow a wide variety of formats, for example:

```
root=/dev/sda1                                 # /dev node
root=LABEL=CorsairF80                          # label
root=UUID=ea1c4959-406c-45d0-a144-912f4e86b207 # UUID
root=PARTUUID=14420948-2cea-4de7-b042-40f67c618660          # GPT partition UUID
```

Note: The following boot parameters alter the default behavior of `init` in the initramfs environment. See `/usr/lib/initcpio/init` for details. They will not work when `systemd` hook is being used since the `init` from `base` hook is replaced.

`break`

If `break` or `break=premount` is specified, `init` pauses the boot process (after loading hooks, but before mounting the root file system) and launches an interactive shell which can be used for troubleshooting purposes. This shell can be launched after the root has been mounted by specifying `break=postmount`. Normal boot continues after exiting from the shell.

`disablehooks`

Disable hooks at runtime by adding `disablehooks=hook1[,hook2,...]`. For example:

```
disablehooks=resume
```

`earlymodules`

Alter the order in which modules are loaded by specifying modules to load early via `earlymodules=mod1[,mod2,...]`. (This may be used, for example, to ensure the correct ordering of multiple network interfaces.)

See **Boot debugging** and **mkinitcpio(8) (https://jlk.fjfi.cvut.cz/arch/manpages/man/mkinitcpio.8)** for other parameters.

Using RAID

First, add the `mdadm_udev` or `mdadm` hook to the `HOOKS` array and any required RAID modules (e.g. raid456, ext4) to the `MODULES` array in `/etc/mkinitcpio.conf`.

Using the `mdadm` hook, you no longer need to configure your RAID array in the **kernel parameters**. The `mdadm` hook will either use your `/etc/mdadm.conf` file or automatically detect the array(s) during the init phase of boot.

Assembly via udev is also possible using the `mdadm_udev` hook. Upstream prefers this method of assembly. `/etc/mdadm.conf` will still be read for purposes of naming the assembled devices if it exists.

Using net

Warning: NFSv4 is not yet supported **FS#28287 (https://bugs.archlinux.org/task/28287)**.

Required Packages

`net` requires the **mkinitcpio-nfs-utils (https://www.archlinux.org/packages/?name=mkinitcpio-nfs-utils)** package.

Kernel Parameters

Comprehensive and up-to-date information can be found in the official **kernel documentation (https://www.kernel.org/doc/Documentation/filesystems/nfs/nfsroot.txt)**.

ip=

This parameter tells the kernel how to configure IP addresses of devices and also how to set up the IP routing table. It can take up to nine arguments separated by colons:

```
ip=<client-ip>:<server-ip>:<gw-ip>:<netmask>:<hostname>:<device>:<autoconf>:<dns0-ip>:<dns1-ip>
```

.

If this parameter is missing from the kernel command line, all fields are assumed to be empty, and the defaults mentioned in the **kernel documentation (https://www.kernel.org/doc/Documentation/filesystems/nfs/nfsroot.txt)** apply. In general this means that the kernel tries to configure everything using autoconfiguration.

The `<autoconf>` parameter can appear alone as the value to the 'ip' parameter (without all the ':' characters before). If the value is `ip=off` or `ip=none`, no autoconfiguration will take place, otherwise autoconfiguration will take place. The most common way to use this is `ip=dhcp`.

For parameters explanation, see the **kernel doc (https://www.kernel.org/doc/Documentation/filesystems/nfs/nfsroot.txt)**.

Examples:

```
ip=127.0.0.1:::::lo:none  --> Enable the loopback interface.
ip=192.168.1.1:::::eth2:none --> Enable static eth2 interface.
ip=:::::eth0:dhcp --> Enable dhcp protocol for eth0 configuration.
```

Note: Make sure to use kernel device names (e.g. `eth0`) for the `<device>` parameter, the persistent names (e.g. `enp2s0`) will not work. See **Network configuration#Network interfaces** for details.

BOOTIF=

If you have multiple network cards, this parameter can include the MAC address of the interface you are booting from. This is often useful as interface numbering may change, or in conjunction with pxelinux IPAPPEND 2 or IPAPPEND 3 option. If not given, eth0 will be used.

Example:

```
BOOTIF=01-A1-B2-C3-D4-E5-F6  # Note the prepended "01-" and capital letters.
```

nfsroot=

If the `nfsroot` parameter is NOT given on the command line, the default `/tftpboot/%s` will be used.

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

Run `mkinitcpio -H net` for parameter explanation.

Using LVM

If your root device is on **LVM**, see **LVM#Configure mkinitcpio**.

Using encrypted root

If using an **encrypted root** see **Dm-crypt/System configuration#mkinitcpio** for detailed information on which hooks to include.

/usr as a separate partition

If you keep `/usr` as a separate partition, you must adhere to the following requirements:

- Add the `fsck` hook, mark `/usr` with a `passno` of `0` in `/etc/fstab`. While recommended for everyone, it is mandatory if you want your `/usr` partition to be fsck'ed at boot-up. Without this hook, `/usr` will never be fsck'd.
- If not using the systemd hook, add the `usr` hook. This will mount the `/usr` partition after root is mounted.

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.)

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.

Standard rescue procedures

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

Boot succeeds on one machine and fails on another

mkinitcpio's `autodetect` hook filters unneeded **kernel modules** in the primary initramfs scanning `/sys` and the modules loaded at the time it is run. If you transfer your `/boot` directory to another machine and the boot sequence fails during early userspace, it may be because the new hardware is not detected due to missing kernel modules. Note that USB 2.0 and 3.0 need different kernel modules.

To fix, first try choosing the **fallback** image from your **bootloader**, as it is not filtered by `autodetect`. Once booted, run *mkinitcpio* on the new machine to rebuild the primary image with the correct modules. If the fallback image fails, try booting into an Arch Linux live CD/USB, chroot into the installation, and run *mkinitcpio* on the new machine. As a last resort, try **manually** adding modules to the initramfs.

See also

- Linux Kernel documentation on **initramfs (https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/plain/Documentation/filesystems/ramfs-rootfs-initramfs.txt?id=HEAD)**

- Linux Kernel documentation on **initrd (https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/plain/Documentation/admin-guide/initrd.rst?id=HEAD)**
- Wikipedia article on **initrd**
- **dracut** — A cross-distribution initramfs generation tool.

  https://dracut.wiki.kernel.org/ || `dracut (https://aur.archlinux.org/packages/dracut/)`AUR

Retrieved from "https://wiki.archlinux.org/index.php?title=Mkinitcpio&oldid=508856"