

initramfs

The initramfs contains all the necessary programs and config files needed to bring up the machine, mount the root file system and hand off the rest of the boot process to the installed system. It can be further customized with additional modules, binaries, files and hooks for special use cases and hardware.

Usage

Automated image generation

Every kernel in Arch Linux comes with its own `.preset` file stored in `/etc/mkinitcpio.d/` with configuration presets for `mkinitcpio`. Pacman hooks build a new image after every kernel upgrade or installation of a new kernel.

Manual image generation

To manually generate a Linux kernel image issue the following command:

```
mkinitcpio -p linux
```

This will generate a new kernel image with the settings of the preset file

```
/etc/mkinitcpio.d/linux.preset
```

.

To generate kernel images with every preset available, pass the `-P` argument:

```
mkinitcpio -P
```

Configuration

To customize your initramfs, place drop-in configuration files into `/etc/mkinitcpio.conf.d/`. They will override the settings in the main configuration file at `/etc/mkinitcpio.conf`.

An overview of the settings you can customize:

Setting	Type	Description
---------	------	-------------

<code>MODULES</code>	Array	Kernel modules to be loaded before any boot hooks are run.
<code>BINARIES</code>	Array	Additional binaries you want included in the initramfs image.
<code>FILES</code>	Array	Additional files you want included in the initramfs image.
<code>HOOKS</code>	Array	Hooks are scripts that execute in the initial ramdisk.
<code>COMPRESSION</code>	String	Which tool to use for compressing the image.
<code>COMPRESSION_OPTIONS</code>	Array	Extra arguments to pass to the <code>COMPRESSION</code> tool.

WARNING: Do not use the `COMPRESSION_OPTIONS` setting, unless you know exactly what you are doing. Misuse can produce unbootable images!

MODULES

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

Here you can specify additional kernel modules needed in early userspace, e.g. file system modules (`ext2`, `reiser4`, `btrfs`), keyboard drivers (`usbhid`, `hid_apple`, etc.), USB 3 hubs (`xhci_hcd`) or "out-of-tree" modules which are not part of the Linux kernel (mainly NVIDIA GPU drivers). It is also needed to add modules for hardware devices that are not always connected but you would like to be operational from the very start if they are connected during boot.

HINT: If you don't know the name of the driver of a device, `lshw` can tell you what hardware uses which driver, e.g.:

```
*-usb:2
    description: USB controller
    product: Tiger Lake-LP USB 3.2 Gen 2x1 xHCI Host Controller
    vendor: Intel Corporation
    physical id: 14
    bus info: pci@0000:00:14.0
    version: 20
    width: 64 bits
    clock: 33MHz
    capabilities: xhci bus_master cap_list
-> configuration: driver=xhci_hcd latency=0
```

```
resources: iomemory:600-5ff irq:163 memory:603f260000-603f26ffff
```

The second to last line starting with `configuration` shows the driver being used.

Example of a `MODULES` array that adds two modules to the generated image needed for keyboard input, if the keyboard is connected to a USB 3 hub, e.g. a docking station:

```
MODULES=(xhci_hcd usbhid)
```

CAUTION: Keep in mind that adding to the `initramfs` increases the size of the resulting image on disk. Unless you have created your boot partition (more specifically the EFI System partition at either `/efi`, `/boot` or `/boot/efi`) with generous space, you should limit yourself to modules strictly needed for your system. The `autodetect` hook tries to detect all currently loaded modules of the running system to determine the needed modules to include by default. Only include additional modules if something doesn't work as expected.

ATTENTION: If you use an NVIDIA graphics card, the following modules are **required** in the `MODULES` array for early KMS:

```
MODULES=(nvidia nvidia_modeset nvidia_uvm nvidia_drm)
```

BINARIES

The `BINARIES` array holds the name of extra executables needed to boot the system. It can also be used to replace binaries provided by `HOOKS`. The executable names are sourced from the `PATH` environment variable, associated libraries are added as well.

Example of a `BINARIES` array that adds the `kexec` binary:

```
BINARIES=(kexec)
```

This option usually only needs to be set for special use cases, e.g. when there's a binary you need included that is not already part of a member in the `HOOKS` array.

FILES

The `FILES` array holds the full path to arbitrary files for inclusion in the image.

Example of a module configuration file to be included in the image, containing the names of modules to auto-load and optional module parameters:

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

This option usually only needs to be set for special use cases.

HOOKS

The `HOOKS` array is the most important setting in the file. Hooks are small scripts which describe what will be added to the image. Hooks are referred to by their name, and executed in the order they are listed in the `HOOKS` array.

HINT: For a full list of available hooks run:

```
mkinitcpio -L
```

See the help text for a hook with:

```
mkinitcpio -H hook_name
```

Alternatively, refer to [Arch Wiki](#) for a complete rundown of all the different hooks and their recommended order.

By default, systemd will bring the whole system up start to finish. In this case bootup will be handled by systemd unit files instead of scripts.

The benefit of this is faster boot times and some additional features like unlocking LUKS encrypted file systems with a TPM or FIDO2 token and automatic detection and mounting of partitions with the appropriate GUID Partition Table (GPT) UUIDs (see: [Discoverable Partition Specification](#)).

The default `HOOKS` array should be enough to bring up most systems. However, if you have special use cases, additional hooks will be needed:

Hook	Description
<code>mdadm_udev</code>	Needed for assembling RAID arrays via udev (software RAID), needs the <code>mdadm</code> package installed
<code>sd-encrypt</code>	Needed for booting from an encrypted file system, needs the <code>cryptsetup</code> package installed
<code>lvm2</code>	Needed for booting a system that is on LVM, needs the <code>lvm2</code> package installed

One such special case is encryption, which would result in a `HOOKS` array that looks like this:

ATTENTION: The order in which hooks are placed in the array is important!

```
HOOKS=(base systemd autodetect microcode modconf kms keyboard sd-vconsole block sd-encrypt
filesystems fsck)
```

ATTENTION: In some cases it might be necessary to place the `keyboard` hook before the `autodetect` hook to be able to enter the passphrase to unlock the encrypted file systems, e.g. when using different keyboards requiring a different module from the one in use at the time of building the initramfs.

COMPRESSION

The `COMPRESSION` option instructs `mkinitcpio` to compress the resulting images to save on space on the EFI System Partition or `/boot` partition. This can be especially important if you include a lot of modules and hooks and the size of the image grows.

Compressing the initramfs is a tradeoff between:

- time it takes to compress the image
- space saved
- time it takes the kernel to decompress the image during boot

Which one you choose is something you have to decide on the constraints you're working with (slow/fast CPU, available cores, RAM usage, disk space), but generally speaking the default `zstd` compression strikes a good balance.

Algorithm	Description
<code>cat</code>	Uncompressed
<code>zstd</code>	Best tradeoff between de-/compression time and image size (default)
<code>gzip</code>	Balanced between speed and size, acceptable performance
<code>bzip2</code>	Rarely used, decent compression, resource conservative
<code>lzma</code>	Very small size, slow to compress
<code>xz</code>	Smallest size at longer compression time, RAM intensive compression
<code>lzop</code>	Slightly better compression than lz4, still fast to decompress

Algorithm	Description
<code>lz4</code>	Fast decompression, slow compression, "largest" compressed output

NOTE: See [this article](#) for a comprehensive comparison between compression algorithms.

COMPRESSION_OPTIONS

WARNING: Misuse of this option may lead to an **unbootable system** if the kernel is unable to unpack the resulting archive. **Do not set** this option unless you're *absolutely* sure that you have to!

The `COMPRESSION_OPTIONS` setting allows you to pass additional parameters for the compression tool. Available parameters depend on the algorithm chosen for the `COMPRESSION` option. Refer to the tool's manual for available options. If left empty `mkinitcpio` will make sure it always produces a working image.

Additionally, `MODULES_DECOMPRESS` instructs `mkinitcpio` to decompress kernel modules prior to inclusion in the initramfs. This can further increase compression efficiency and bring down the initramfs size further. When this option is not set, compressed kernel modules are included as-is.

For example, to use the maximum zstd compression level, using all available CPU cores and show verbose output during compression:

```
COMPRESSION="zstd"
COMPRESSION_OPTIONS=(-T0 -19 --long --auto-threads=logical -v)
MODULES_DECOMPRESS="yes"
```

Unified Kernel Image

A unified kernel image (UKI) combines an EFI stub image, CPU microcode, kernel command line and an initramfs into a single file that can be read and executed by the machine's UEFI firmware, thus making a boot manager potentially redundant. Additionally, it streamlines the process of signing for secure boot, as there is only a single file to sign.

Version 31 of `mkinitcpio` introduced support for building UKIs out of the box. Starting with v39, `systemd-ukify` is the recommended method by which to generate UKIs. As `systemd-ukify` is *not* part of the `systemd` package, you'll have to install it manually:

```
pacman -S systemd-ukify
```

To make `mkinitcpio` generate UKIs, edit the appropriate `*.preset` file for your kernel in `/etc/mkinitcpio.d/`:

- comment out the `default_image` and `fallback_image` lines (as they won't be needed)
- uncomment the `default_uki` and `fallback_uki` lines (prompts `mkinitcpio` to switch to UKI generation)
- point the file path to somewhere on your EFI System Partition (e.g. `/efi`)

NOTE: `mkinitcpio` will automatically source command line parameters from files in `/etc/cmdline.d/*.conf` or a complete single command line from `/etc/kernel/cmdline`. If you need different images to use different kernel command line parameters, the `*_options` line in the `*.preset` allows you to pass additional arguments to `mkinitcpio`, i.e. the `--cmdline` argument to point it to a different file containing a different set of kernel command line parameters.

NOTE: Placing the UKI under `/efi/EFI/Linux/` allows `systemd-boot` to automatically detect images and list them without having to specifically create boot entries for them.

A `*.preset` file edited for UKI generation could look something like this:

```
# mkinitcpio preset file for the 'linux' package

#ALL_config="/etc/mkinitcpio.conf"
ALL_kver="/boot/vmlinuz-linux"
#ALL_kerneldest="/boot/vmlinuz-linux"

#PRESETS=('default')
PRESETS=('default' 'fallback')

#default_config="/etc/mkinitcpio.conf"
#default_image="/boot/initramfs-linux.img"
default_uki="/efi/EFI/Linux/arch-linux.efi"
#default_options="--splash /usr/share/systemd/bootctl/splash-arch.bmp"

#fallback_config="/etc/mkinitcpio.conf"
#fallback_image="/boot/initramfs-linux-fallback.img"
fallback_uki="/efi/EFI/Linux/arch-linux-fallback.efi"
fallback_options="-S autodetect,plymouth --cmdline /etc/kernel/cmdline_fallback"
```

This `*.preset` file instructs `mkinitcpio` to generate a UKI and enables the fallback initramfs. It skips the `autodetect` and `plymouth` hooks for the fallback initramfs and passes a different set of kernel

command line parameters, e.g. displaying boot logs instead of showing a splash screen.

Kernel Command Line Parameters

`mkinitcpio` automatically looks for kernel command line parameters specified in `/etc/cmdline.d/*.conf` as drop-in files or `/etc/kernel/cmdline` as a single file.

WARNING: If `mkinitcpio` does not find command line parameters in either of the above locations, it will fall back to reading the command line of the currently booted system from `/proc/cmdline`. If you're booted into the Arch installation environment, this will most likely leave you with an unbootable system. **Set at least one command line parameter in one of the above locations!**

Create the directory for command line parameter drop-in files and start with specifying parameters for the root file system:

```
mkdir /etc/cmdline.d
nano /etc/cmdline.d/root.conf
```

Continue by specifying the root file system via [persistent block device naming](#) and mounting it writable:

```
root=UUID=XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX rw
```

You can add as many `*.conf` files as you need to logically split up kernel parameters. All of the parameters from all files will be included in the UKI.

GPT auto mounting

Since the default `initramfs` type is `systemd`-based, it's possible to rely on [systemd-gpt-auto-generator\(8\)](#) for automatic discovery and mounting of file systems during boot.

ATTENTION: This requires that the correct GPT partition types were set during partitioning and that the file systems to be auto mounted are located on the same disk as the EFI system partition. If other important file systems are located on other disks, they must still be specified via `/etc/crypttab` and `/etc/fstab`.

GPT auto mounting will create a symbolic link to the root file system and the encrypted file system by which it can be addressed. This can be specified in a file like `/etc/crypttab.initramfs` to be included at boot time (see [crypttab\(5\)](#) for details on the syntax):

NOTE: By default, dm-crypt does not allow TRIM for SSDs for security reasons (information leak). To override this behavior, either specify `rd.luks.options=discard` as an additional kernel command line parameter or add the `discard` option in `/etc/crypttab.initramfs` in the options field.

```
# <name>      <device>                                <passphrase>  <options>
root          /dev/gpt-auto-root-luks
```

During boot, the system will ask for the passphrase for the encrypted file system and systemd will mount the unlocked filesystem automatically. If there are additional options you would like to pass, specify them as additional parameters in the `<options>` column.

With this type of configuration, `root` and `rd.luks` can be omitted entirely from the required list of kernel command line parameters:

ATTENTION: Be aware of the specifics of your chosen root file system. For example, when using btrfs, you will still need to specify the subvolume and any other file system options as kernel command line parameters, as automatic discovery and mounting will use the default options for mounting file systems: `rootflags=noatime,compress=zstd,subvol=@`.

```
rw
```

Once at least the root file system has been mounted, the boot process continues to mount file systems specified via `/etc/crypttab` and `/etc/fstab` like normal.

Manually

In cases where GPT auto mounting is not possible or undesired, the manual way of specifying encrypted devices remains available.

In a systemd-based initramfs, `rd.luks.name` is used to specify the encrypted partition by its UUID and a mapper name by which the decrypted file system is made available, resulting in a kernel command line that looks like this:

NOTE: By default, dm-crypt does not allow TRIM for SSDs for security reasons (information leak). To override this behavior, either specify `rd.luks.options=discard` as an additional kernel command line parameter or add the `discard` option in `/etc/crypttab.initramfs` in the options field.

```
rd.luks.name=XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX=root root=/dev/mapper/root rw
```

The UUID of the encrypted file system can be determined using `blkid` (assuming `/dev/nvme0n1p3` is the encrypted file system):

NOTE: Pressing `Ctrl + T` inside `nano` allows you to paste the result of a command at the current cursor position.

```
blkid -s UUID -o value /dev/nvme0n1p3
```

If you prefer a config file approach, or need to mount multiple encrypted file systems during boot, the same `/etc/crypttab` file can be used to specify all encrypted devices. Using [persistent block device naming](#), the file could look like this (see [crypttab\(5\)](#) for details on the syntax)::

```
# <name>      <device>                                <passphrase>  <options>
root          UUID=XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
```

This allows for omitting any `rd.luks` parameters entirely:

ATTENTION: Be aware of the specifics of your chosen root file system. For example, when using `btrfs`, you will still need to specify the subvolume and any other file system options as kernel command line parameters, as automatic discovery and mounting will use the default options for mounting file systems: `rootflags=noatime,compress=zstd,subvol=@`.

```
root=/dev/mapper/root rw
```

Revision #16

Created 2022-02-12 00:58:09 UTC by Sebin

Updated 2026-02-07 02:18:49 UTC by Sebin