Boot Loader

`systemd-boot`

`systemd` comes with `systemd-boot` already, so no additional packages need to be installed.

Install

> **ATTENTION:** By default, `systemd-boot` will install itself to either of the well-known ESP locations, e.g. `/efi`, `/boot`, or (discouraged) `/boot/efi`. If your ESP is mounted somewhere else pass the localtion with the `--esp-path` parameter. `$ESP` refers to this location. Adjust paths accordingly!

> **WARNING:** `bootctl` will not operate on UEFI variables which store boot entries when running in regular `arch-chroot`, which could leave your machine unbootable. Enter a chroot environement with `arch-chroot -S` instead.

Install `systemd-boot` by simply invoking `bootctl` with the `install` command:

```
bootctl install
```

This will do the following:

- Create the directory `$ESP/EFI/Linux`
- Copy `/usr/lib/systemd/boot/efi/systemd-bootx64.efi` to `$ESP/EFI/systemd/systemd-bootx64.efi`
- Copy `/usr/lib/systemd/boot/efi/systemd-bootx64.efi` to `$ESP/EFI/BOOT/BOOTX64.EFI`
- Create a 32 byte random seed file at `$ESP/loader/random-seed`
- Create an EFI boot entry named *Linux Boot Manager* at the top of firmware boot entries

> **NOTE:** If a signed version of `systemd-bootx64.efi` exists as `systemd-bootx64.efi.signed` in the source directory (i.e. for Secure Boot), the signed file is copied instead.

> **NOTE:** `bootctl` may complain about your ESP's mount point and the random seed file as being "world accessible". This is to let you know your ESP's current file system permissions

> are too lenient. To solve this, change the `fmask` and `dmask` mount options for your ESP in `/etc/fstab` from `0022` to `0077`. Changes apply on next boot. See also: `mount(8) $ Mount options for fat`. If you plan on using `systemd`'s GPT auto-mounting feature, it will set the appropriate file system permissions for you.

Configure

`systemd-boot` has two kinds of configs:

- `$ESP/loader/loader.conf`: Configuration file for the boot loader itself
- `$ESP/loader/entries/*.conf`: Configuration files for individual boot entries

Boot loader config

> **NOTE:** For a full list of options and their explanation refer to `loader.conf(5) § OPTIONS`

The most important options for the boot loader are as follows:

Setting	Type	Description
`default`	string	The pre-selected default boot entry. Can be pre-determined value, file name or glob pattern
`timeout`	number	Time in seconds until the default entry is automatically booted
`console-mode`	number/string	Display resolution mode (`0`, `1`, `2`, `auto`, `max`, `keep`)
`auto-entries`	number/boolean	Show/hide other boot entries found by scanning the boot partition
`auto-firmware`	number/boolean	Show/hide "Reboot into firmware" entry

An example loader configuration could look something like this:

> **ATTENTION:** Only spaces are accepted as white-space characters for indentation, do not use tabs!

```
default         arch    # pre-selects entry from $ESP/loader/entries/arch.conf
timeout         3       # 3 seconds before the default entry is booted
auto-entries    1       # shows boot entries which were auto-detected
auto-firmware   1       # shows entry "Reboot into firmware"
console-mode    max     # picks the highest-numbered mode available
```

Boot entry config

> **SEE ALSO:** The Boot Loader Specification for a comprehensive overview of what `systemd-boot` implements.

Available parameters in boot entry config files:

Key	Value	Description
`title`	string	The name of the entry in the boot menu (optional)
`version`	string	Human readable version of the entry (optional)
`machine-id`	string	The unique machine ID of the computer (optional)
`sort-key`	string	Used for sorting entries (optional)
`linux`	path	Location of the Linux kernel (relative to ESP)
`initrd`	path	Location of the Linux initrd image (relative to ESP)
`efi`	path	Location of an EFI executable, hidden on non-EFI systems
`options`	string	Kernel command line parameters
`devicetree`	path	Binary device tree to use when executing the kernel (optional)
`devicetree-overlay`	paths	List of device tree overlays. If multiple, separate by space, applied in order
`architecture`	string	Architecture the entry is intended for (`IA32` , `x64` , `ARM` , `AA64`)

Type 1 (text file based)

> **NOTE:** As of mkinitramfs v38, the CPU microcode is embedded in the initramfs and it is no longer necessary to specify CPU microcode images on a separate initrd line before the actual initramfs.

Type 1 entries specify their parameters in `*.conf` files under `§ESP/loader/entries/`.

All paths in these configs are relative to the ESP, e.g. if the ESP is mounted at `/boot` a boot loader entry located at `$ESP/loader/entries/arch.conf` would look like this:

```
title Arch Linux
linux /vmlinuz-linux
initrd /initramfs-linux.img
options rd.luks.name=XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX=cryptroot root=/dev/mapper/cryptroot
rw
```

Type 2 (EFI executable)

When using a unified kernel image, any image ending with `*.efi` placed under `$ESP/EFI/Linux/` will be automatically picked up by `systemd-boot` along with the metadata embedded in that image (e.g. title, version, etc.)

If your UKIs are stored somewhere else, you will need a loader entry `*.conf` file with an `efi` key pointing `systemd-boot` to the location of the `*.efi` file on the ESP:

```
title Arch Linux
efi     /EFI/Arch/linux.efi
```

EFISTUB

EFISTUB is a method of booting the kernel directly as an EFI executable by the firmware without the need for a boot loader. This can be useful in cases where you want to reduce the attack surface a boot loader can introduce, or you intend to only ever boot one image. However, some UEFI firmware implementations can be flaky, so this isn't always practical.

Install

To be able to manipulate EFI boot variables install `efibootmgr`:

```
pacman -S efibootmgr
```

Configure

> **ATTENTION:** `efibootmgr` cannot overwrite existing boot entries and will disregard the creation of a boot entry if one with the same label already exists. If you need to overwrite an existing entry you will need to delete it first. Call `efibootmgr` without any arguments to list all current boot entries:
>
> ```
> efibootmgr
> ```

To delete an entry, note its 4-digit boot entry order and instruct `efibootmgr` to delete it:

```
efibootmgr -Bb XXXX
```

To create a new entry `efibootmgr` needs to know the disk and partition where the kernel image resides on the ESP.

In this example, the ESP is the first partition of the block device `/dev/nvme0n1`. Kernel parameters are part of the `-u` option. The partition that holds your root file system needs to be passed as a **persistent block device name**.

NOTE: If you use LVM or LUKS, you can skip this step and supply the device mapper name since that already is persistent.

You can get the persistent block device identifier of a file system with the `blkid` command, i.e. to get the UUID of the root file system. For example, if `/dev/nvme0n1p2` is the root file system:

```
blkid -s UUID -o value /dev/nvme0n1p2
```

For ease of scriptability, save the values to environment variables:

```
export ROOT=$(blkid -s UUID -o value /dev/nvme0n1p2)
export CMDL="root=UUID=$ROOT rw add_efi_memmap initrd=\\\initramfs-linux.img"
```

Then create the boot entry using `efibootmgr`:

```
efibootmgr -c -L "Arch Linux" -d /dev/nvme0n1 -p 1 -l /vmlinuz-linux -u $CMDL -v
```

Unified kernel image

When using a **unified kernel image** you can instead just point to the UKI without needing to specify any kernel parameters via the `-u` option (as these will be part of the UKI already):

ATTENTION: If Secure Boot is enabled and the command line parameters are embedded in the UKI, the embedded command line parameters will always take precedence, even if you pass additional parameters with the `-u` option.

```
efibootmgr -c -L "Arch Linux" -d /dev/nvme0n1 -p 1 -l "EFI\Linux\archlinux-linux.efi" -v
```

Revision #18
Created 2021-04-03 22:19:38 UTC by Sebin
Updated 2026-02-06 17:58:17 UTC by Sebin