cmkinitramfs

Release 0.2.2

Louis Leseur

Nov 04, 2022

CONTENTS

Tools to generate an initramfs from a configuration file.

Documentation is available at https://cmkinitramfs.readthedocs.io/.

CHAPTER

ONE

ABOUT

This project provides three main executables: `cmkinit`, `cmkcpiodir` and `cmkcpiolist`.

`cmkinit` builds an init script from a configuration file.

`cmkcpiodir` and `cmkcpiolist` build an initramfs, including the init script, from the same configuration file. `cmkcpiodir` builds the initramfs into a directory on a filesystem, and generates the CPIO archive from it. `cmkcpiolist` builds a CPIO list, using the same format as Linux kernel's `gen_init_cpio` utility, and generates the CPIO archive using `gen_init_cpio`. See the corresponding Linux kernel documentation for more information.

CHAPTER

TWO

INSTALLATION

2.1 Compatibility

Python version: this library is compatible with **Python 3.7**.

Python implementation: this library is compatible with **CPython** and **PyPy**.

2.2 Dependencies

Python dependencies:

- bin (mkcpiodir and mkcpiolist) dependencies:
  - `pyelftools`
- Documentation:
  - `sphinx`
  - `sphinx_rtd_theme`
- Tests:
  - QA:
    - `flake8`
    - `mypy`
    - `tox`

Other dependencies:

- initramfs (mkcpiodir and mkcpiolist) dependencies:
  - `loadkeys` (kbd)
  - `busybox`
  - `modinfo` (kmod, busybox)
- mkcpiodir dependencies:
  - `find` (findutils, busybox)
  - `cpio` (cpio, busybox)
- mkcpiolist dependencies:
  - `gen_init_cpio` (linux kernel, linux-misc-apps)

2.3 Install

Install from pypi:

```
$ pip install cmkinitramfs
```

Install from source with setup.py:

```
$ git clone https://github.com/teapot9/cmkinitramfs.git
$ cd cmkinitramfs
$ python3 setup.py install
```

Install from source with pip:

```
$ git clone https://github.com/teapot9/cmkinitramfs.git
$ cd cmkinitramfs
$ pip3 install .
```

CHAPTER

THREE

CONFIGURATION

The configuration file is in an *ini* format.

Each section defines a data source, the section name is the data identifier.

Some options expects a data source as input, there are several data identifier formats:

- `DATA=data-name`: data defined in the section with the same name.
- `data-name`: same as `DATA=data-name`.
- `PATH=/path/foo/bar`: data at the path `/path/foo/bar`, this can be a directory, a file, or a block device.
- `/absolute/path`: same as `PATH=/absolute/path`.
- `UUID=1234-5678`: filesystem with UUID `1234-5678`.
- `LABEL=foo`: filesystem with label `foo`.
- `PARTUUID=1234-5678`: partition with UUID `1234-5678`.
- `PARTLABEL=foo`: partition with label `foo`.

3.1 DEFAULT section

This section has default values for other sections, as well as global configuration.

- `root` (mandatory): Data identifier for the data to use as new root.
- `mountpoints` (optional): Comma separated list of data identifier to load in addition of rootfs.
- `keymap` (optional): Boolean value defining if a keymap should be loaded. If set to `no`, all `keymap-*` configurations will be ignored. Defaults to `no`.
- `keymap-src` (optional): Path of the keymap file to use. If not specified but `keymap` is `yes`, the converted keymap should already exists at `keymap-path`.
- `keymap-path` (optional): Path where the binary keymap will be generated (generated from `keymap-src`). Defaults to `/tmp/keymap.bmap`.
- `keymap-dest` (optional): Path of the keymap file within the initramfs. Defaults to `/root/keymap.bmap`.
- `init-path` (optional): Path where the init script will be generated (generated from `cmkinitramfs.init.mkinit()`). Defaults to `/tmp/init.sh`.
- `files` (optional): Additional files to include in the initramfs. Each item is separated by a newline. Format: `source:destination` (e.g. `files = /root/foo:/root/bar` copy the file `foo` in the initramfs renaming it `bar`). If no destination is given, the file will be copied to the same path as `source` in the initramfs. `source` can be an absolute or relative path, `destination` must be an absolute path within the initramfs.

- `execs` (optional): Additional executables to include in the initramfs. Same format as `files`, except that `source` will also be searched in directories from the `PATH` environment variable.
- `libs` (optional): Additional libraries to include in the initramfs. Same format as `files`, except that `source` will also be searched in directories from `/etc/ld.so.conf` and the `LD_LIBRARY_PATH` environment variable.
- `busybox` (optional): Additional executables to include in the initramfs. Each item is separated by a newline. Format: `exec`: name of the command (basename). If busybox provides the command, they will not be added. Otherwise, the executable is searched in `PATH`.
- `cmkcpiodir-default-opts` (optional): Options to append to the `cmkcpiodir` command line.
- `cmkcpiolist-default-opts` (optional): Options to append to the `cmkcpiolist` command line.
- `modules` (optional): Kernel modules to load in the initramfs. One module per line, each line with the module name followed by the module parameters (e.g. `mymodule foo=bar`).
- `scripts` (optional): User scripts to run at a given breakpoint. One user script per line with the format `breakpoint:script`. The script `script` will be run at the breakpoint `breakpoint`. A list of available breakpoints is available in `cmkinitramfs.init.Breakpoint`. These scripts will be run wether the breakpoint is enabled or not. Example: `init: ls /dev`: run `ls /dev` after initialization.

3.2 LUKS data sections

LUKS device to open.

- `type = luks` (mandatory).
- `need` (optional): Hard dependencies: comma separated list of data identifiers. Those dependencies are required to load *and* use the data.
- `load-need` (optional): Load dependencies: comma separated list of data identifiers. Those dependencies are only required to load the data, they can be unloaded when the data has been successfully loaded. (e.g. A LUKS key, an archive to decompress.)
- `source` (mandatory): Data identifier of the data to unlock.
- `name` (mandatory): Name to use for the luks device, this will be used by cryptsetup.
- `key` (optional): Data identifier for the LUKS key.
- `header` (optional): Data identifier for the LUKS header.
- `discard` (optional): Enable discards. Boolean value (yes/no).

3.3 LVM data sections

LVM logical volume to load.

- `type = lvm` (mandatory).
- `need` (optional): Hard dependencies: comma separated list of data identifiers. Those dependencies are required to load *and* use the data.
- `load-need` (optional): Load dependencies: comma separated list of data identifiers. Those dependencies are only required to load the data, they can be unloaded when the data has been successfully loaded. (e.g. A LUKS key, an archive to decompress.)
- `vg-name` (mandatory): Volume group name.

- `lv-name` (mandatory): Logical volume name.

3.4 Mount data sections

Filesystem to mount.

- `type = mount` (mandatory).
- `need` (optional): Hard dependencies: comma separated list of data identifiers. Those dependencies are required to load *and* use the data.
- `load-need` (optional): Load dependencies: comma separated list of data identifiers. Those dependencies are only required to load the data, they can be unloaded when the data has been successfully loaded. (e.g. A LUKS key, an archive to decompress.)
- `source` (optional): Data identifier for the filesystem to mount. If not set, it will set the source to "none" (e.g. for TMPFS).
- `mountpoint` (mandatory): Path where the filesystem will be mounted.
- `filesystem` (mandatory): Which filesystem to use, option passed to `mount -t filesystem`.
- `options` (optional): Mount options, defaults to `ro`. Note for ZFS: if the `mountpoint` property is not set to `legacy`, the `zfsutil` option is required.

3.5 MD data sections

MD RAID data to load.

- `type = md` (mandatory).
- `need` (optional): Hard dependencies: comma separated list of data identifiers. Those dependencies are required to load *and* use the data.
- `load-need` (optional): Load dependencies: comma separated list of data identifiers. Those dependencies are only required to load the data, they can be unloaded when the data has been successfully loaded. (e.g. A LUKS key, an archive to decompress.)
- `name` (mandatory): Name of the MD RAID, this will be used by mdadm.
- `source` (mandatory): New line separated data identifiers of the sources to use. Multiple block devices can be specified, or the UUID of the MD RAID.

3.6 Clone data sections

Clone a source to a destination.

- `type = clone` (mandatory).
- `need` (optional): Hard dependencies: comma separated list of data identifiers. Those dependencies are required to load *and* use the data.
- `load-need` (optional): Load dependencies: comma separated list of data identifiers. Those dependencies are only required to load the data, they can be unloaded when the data has been successfully loaded. (e.g. A LUKS key, an archive to decompress.)
- `source` (mandatory): Data identifier for the source of the clone.

- `destination` (mandatory): Data identifier of the destination of the clone.

3.7 ZFS pool sections

Import a ZFS pool.

- `type = zfspool` (mandatory).
- `need` (optional): Hard dependencies: comma separated list of data identifiers. Those dependencies are required to load *and* use the data.
- `load-need` (optional): Load dependencies: comma separated list of data identifiers. Those dependencies are only required to load the data, they can be unloaded when the data has been successfully loaded. (e.g. A LUKS key, an archive to decompress.)
- `pool` (mandatory): Pool name.
- `cache` (optional): Data identifier of the ZFS cache file (must be present in the initramfs).

3.8 ZFS crypt sections

Unlock an encrypted ZFS dataset.

- `type = zfscrypt` (mandatory).
- `need` (optional): Hard dependencies: comma separated list of data identifiers. Those dependencies are required to load *and* use the data.
- `load-need` (optional): Load dependencies: comma separated list of data identifiers. Those dependencies are only required to load the data, they can be unloaded when the data has been successfully loaded. (e.g. A LUKS key, an archive to decompress.)
- `pool` (optional): Data identifier of the parent pool (defaults to same as pool name).
- `dataset` (mandatory): Name of the dataset to unlock.
- `key` (optional): Data identifier of a keyfile to use.

CHAPTER

FOUR

USAGE

4.1 Kernel command-line parameters

The init script will check the kernel cmdline for known parameters.

- `debug`: Same as `rd.debug`.
- `init=<path to init>`: Set the init process to run after the initramfs.
- `quiet`: Same as `rd.quiet`.
- `rd.break=<breakpoint>`: Drop into a shell at a given point. See `cmkinitramfs.init.Breakpoint`.
- `rd.debug`: Show debugging informations.
- `rd.panic`: On fatal error: cause a kernel panic rather than droping into a shell.
- `rd.quiet`: Reduce log shown on console.

For more details, see `cmkinitramfs.init.do_cmdline`.

4.2 cmkinit

```
$ cmkinit --help
usage: cmkinit [-h] [--version]

Build an init script

optional arguments:
  -h, --help  show this help message and exit
  --version   show program's version number and exit
```

Running `cmkinit` will generate an init script and output it to stdout. No options are available, everything is defined in the configuration file. The `CMKINITCFG` environment variable may be defined to use a custom configuration file.

4.3 cmkcpiodir

```
$ cmkcpiodir --help
usage: cmkcpiodir [-h] [--version] [--debug] [--verbose] [--quiet]
                  [--output OUTPUT] [--binroot BINROOT] [--kernel KERNEL]
                  [--only-build-archive | --only-build-directory] [--keep]
                  [--clean] [--build-dir BUILD_DIR]

Build an initramfs using a directory.

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  --debug, -d           debugging mode: non-root, implies -k
  --verbose, -v         be verbose
  --quiet, -q           be quiet (can be repeated)
  --output OUTPUT, -o OUTPUT
                        set the output of the CPIO archive
  --binroot BINROOT, -r BINROOT
                        set the root directory for binaries (executables and
                        libraries)
  --kernel KERNEL, -K KERNEL
                        set the target kernel version of the initramfs,
                        defaults to the running kernel
  --only-build-archive, -c
                        only build the CPIO archive from an existing initramfs
                        directory
  --only-build-directory, -D
                        only build the initramfs directory, implies -k
  --keep, -k            keep the created initramfs directory
  --clean, -C           overwrite temporary directory if it exists, use
                        carefully
  --build-dir BUILD_DIR, -b BUILD_DIR
                        set the location of the initramfs directory
```

Running `cmkcpiodir` will generate the initramfs in a directory, then it will create the CPIO archive from this directory. `cmkcpiodir` requires root privileges when run in non-debug mode, see the `do_nodes` options of `cmkinitramfs.initramfs.Initramfs.build_to_directory()`.

4.4 cmkcpiolist

```
$ cmkcpiolist --help
usage: cmkcpiolist [-h] [--version] [--debug] [--verbose] [--quiet]
                   [--output OUTPUT] [--binroot BINROOT] [--kernel KERNEL]
                   [--only-build-archive | --only-build-list] [--keep]
                   [--cpio-list CPIO_LIST]

Build an initramfs using a CPIO list

optional arguments:
```

(continues on next page)

(continued from previous page)

```
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  --debug, -d           debugging mode: non-root, implies -k
  --verbose, -v         be verbose
  --quiet, -q           be quiet (can be repeated)
  --output OUTPUT, -o OUTPUT
                        set the output of the CPIO archive
  --binroot BINROOT, -r BINROOT
                        set the root directory for binaries (executables and
                        libraries)
  --kernel KERNEL, -K KERNEL
                        set the target kernel version of the initramfs,
                        defaults to the running kernel
  --only-build-archive, -c
                        only build the CPIO archive from an existing CPIO list
  --only-build-list, -L
                        only build the CPIO list, implies -k
  --keep, -k            keep the created CPIO list
  --cpio-list CPIO_LIST, -l CPIO_LIST
                        set the location of the CPIO list
```

Running `cmkcpiolist` will generate an initramfs CPIO list in a file, then it will create the CPIO archive from this list with `gen_init_cpio`. `cmkcpiolist` does not require root privileges.

4.5 findlib

```
$ findlib --help
usage: findlib [-h] [--verbose] [--quiet] [--version]
               [--compatible COMPATIBLE] [--root ROOT] [--null] [--glob]
               LIB [LIB ...]

Find a library on the system

positional arguments:
  LIB                   library to search

optional arguments:
  -h, --help            show this help message and exit
  --verbose, -v         be verbose
  --quiet, -q           be quiet (can be repeated)
  --version             show program's version number and exit
  --compatible COMPATIBLE, -c COMPATIBLE
                        set a binary the library must be compatible with
  --root ROOT, -r ROOT  set the root directory to search for the library
  --null, -0            paths will be delemited by null characters instead of
                        newlines
  --glob, -g            library names are glob patterns
```

`findlib` will search the absolute path of a library on the system. It will search in directories from `/etc/ld.so.conf`, `LD_LIBRARY_PATH`, and default library paths (see `cmkinitramfs.bin.find_lib()` and `cmkinitramfs.bin.find_lib_iter()`).

CHAPTER

FIVE

EXAMPLES

5.1 Command-line interface

```
$ cmkcpiodir
```

- Creates init script in `/tmp/init.sh`.
- If enabled, builds binary keymap in `/tmp/keymap.bmap`.
- Builds initramfs in `/tmp/initramfs` (disable this step with `--only-build-archive`).
- Builds CPIO archive from `/tmp/initramfs` to `/usr/src/initramfs.cpio` (disable this step with `--only-build-directory`).
- Cleanup `/tmp/initramfs` directory (disable with `--keep`).

```
$ cmkcpiolist
```

- Creates init script in `/tmp/init.sh`.
- If enabled, builds binary keymap in `/tmp/keymap.bmap`.
- Builds CPIO list in `/tmp/initramfs.list` (disable this step with `--only-build-archive`).
- Builds CPIO archive from `/tmp/initramfs.list` to `/usr/src/initramfs.cpio` (disable this step with `--only-build-list`).

```
$ findlib 'libgcc_s.so.1'
/usr/lib/gcc/x86_64-pc-linux-gnu/10.2.0/libgcc_s.so.1
```

- Searches the `libgcc_s.so.1` library on the system and prints it to stdout.

```
$ findlib -g 'libgcc_s.*'
/usr/lib/gcc/x86_64-pc-linux-gnu/10.2.0/libgcc_s.so.1
/lib64/libgcc_s.so.1
/lib64/libgcc_s.so.1
```

- Search any library matching `libgcc_s.*` on the system and prints them to stdout.

5.2 Configuration

```
# cmkinitramfs.ini
#
# Example configuration file for cmkinitramfs and cmkinit
#
# This configuration uses a separate / and /usr partitions, respectively on
# system/root and system/usr LVM logical volumes. The LVM is on a LUKS
# encrypted partition. The key to unlock the LUKS partition is encrypted
# in the /root/key file, the header for the key file is /root/key.luks.
# The luks encrypted partition is on a MD RAID.
#
# The resulting /init script will:
#   - Assemble the RAID
#   - Unlock the key /root/key with the header /root/key.luks
#   - Unlock the main luks partition with the unlocked key
#   - Lock the key since we don't need it anymore
#   - Enable the system/root LV
#   - Mount /
#   - User script: print the content of /dev
#   - Enable the system/usr LV
#   - Mount /usr
#

[DEFAULT]
root = mnt-root
mountpoints = mnt-usr
keymap = yes
keymap-src = fr
#keymap-dest = /root/keymap.bmap
init = /sbin/openrc-init
files = /root/key:/root/key.luks
scripts =
        rootfs: echo 'Content of /dev:'
        rootfs: ls /dev

[md-main]
type = md
need =
load-need =
name = mdraid
source = UUID=xxxxxxxx-yyyy-yyyy-yyyy-zzzzzzzzzzzz

[luks-key]
type = luks
need =
load-need =
source = PATH=/root/key
name = key
header = PATH=/root/key.luks

[luks-main]
```

(continues on next page)

(continued from previous page)

```
type = luks
need = md-main
load-need = luks-key
source = UUID=xxxxxxxx-yyyy-yyyy-yyyy-zzzzzzzzzzzz
name = luks
key = luks-key
discard = yes

[lvm-root]
type = lvm
need = luks-main
load-need =
vg-name = system
lv-name = root

[mnt-root]
type = mount
need = lvm-root
load-need =
source = UUID=xxxxxxxx-yyyy-yyyy-yyyy-zzzzzzzzzzzz
mountpoint = /mnt/root
filesystem = ext4
options = ro

[lvm-usr]
type = lvm
need = luks-main
load-need =
vg-name = system
lv-name = usr

[mnt-usr]
type = mount
need = lvm-usr
load-need =
source = lvm-usr
mountpoint = /mnt/root/usr
filesystem = ext4
options = ro
```

CHAPTER

SIX

TABLE OF CONTENTS

6.1 init

Module providing functions to build an init script

The *mkinit()* function will generate a `/init` script.

`do_foo()` functions write a string performing the foo action into a stream. This stream should be the init script.

`_fun_foo()` functions write a string defining the foo function into a stream. This stream should be the init script.

Helper functions available in the init script that can be used by *cmkinitramfs.data.Data* classes:

- *_fun_die()*: Fatal error handler (does not return).
- *_fun_log()*: Logging functions (always successful).

Special helper function available in the init script: *_fun_rescue_shell()*, *_fun_panic()*.

Init environment variables:

- `HOME`
- `PATH`

Init global variables:

- `PRINTK`: Initial kernel log level.
- `INIT`: Program to run as init process after the initramfs.
- `RD_XXX`: If set, feature *XXX* is enabled.
- `RD_BREAK_XXX`: If set, breakpoint *XXX* is enabled.

The init script should never rely on global variables being set or unset, it should always assign a default value if not set.

class cmkinitramfs.init.**Breakpoint**(*value*)

Bases: Enum

Breakpoint in the boot process

Breakpoints can be enabled by adding rd.break to the kernel command-line (e.g. `./kernel.img foo rd.break=init`). Setting `rd.break=foo,bar` will enable both `foo` and `bar`. Environment variables can also be set to enable them (e.g. `./kernel.img foo RD_BREAK_EARLY=true`).

EARLY = 1

Early break: break before any action, including command-line parsing. Can be set with the `RD_BREAK_EARLY` environment variable.

INIT = 2

`init`: Break after initramfs initialization. Can also be set with the `RD_BREAK_INIT` environment variable.

MODULE = 3

`module`: Break after loading kernel modules. Can also be set with the `RD_BREAK_MODULE` environment variable. Alias: `modules`.

MOUNT = 5

`mount`: Break after mounting all filesystems. Can also be set with the `RD_BREAK_MOUNT` environment variable. Alias: `mounts`.

ROOTFS = 4

`rootfs`: Break after mounting the root filesystem. Can also be set with the `RD_BREAK_ROOTFS` environment variable.

cmkinitramfs.init.**_fun_rescue_shell**(*out*)

Define the rescue_shell function

`rescue_shell` drop the user to `/bin/sh`.

Arguments: none.

This function *should not* be called from a subshell.

This function *does not* return.

Parameters
out (*IO[str]*) – Stream to write into

Return type
None

cmkinitramfs.init.**_fun_panic**(*out*)

Define the panic function

`panic` causes a kernel panic by exiting `/init`.

Arguments: none.

This function *should not* be called from a subshell.

This function *does not* return.

Parameters
out (*IO[str]*) – Stream to write into

Return type
None

cmkinitramfs.init.**_fun_die**(*out*)

Define the die function

`die` will either start a rescue shell or cause a kernel panic, wether `RD_PANIC` is set or not.

Arguments: error message.

This function *should not* be called from a subshell.

This function *does not* return.

Parameters
out (*IO[str]*) – Stream to write into

Return type
None

cmkinitramfs.init._fun_log(*out*)

Define the logging functions

`log`: log a message.

- Argument 1: syslog level number, from 0 to 7.
- Additionnal arguments: message to log.

Logs printed to stderr:

- Level 4: always
- 5 level 6: if debug enabled or quiet disabled
- Level = 7: if debug enabled

Helper functions:

- `emerg`: log a message for a panic condition. The message is prepended by 'FATAL:'.
- `alert`: log a critical error message requiring immediate action. The message is prepended by 'ERROR:'.
- `crit`: log a critical error message. The message is prepended by 'ERROR:'.
- `err`: log an error message. The message is prepended by 'ERROR:'.
- `warn`: log a warning message. The message is prepended by 'WARNING:'.
- `notice`: log a significant/unusual informational message.
- `info`: log an informational message.
- `debug`: log a debug-level message.

Helper functions will call `log` with the coresponding syslog level.

Logging functions always return successfully.

Parameters
out (*IO[str]*) – Stream to write into

Return type
None

cmkinitramfs.init.do_header(*out*, *home='/root'*, *path='/bin:/sbin'*)

Create the /init header

- Create the shebang `/bin/sh`
- Configure environment variables
- Define global functions (`panic`, logging, ...)

Parameters

- **out** (*IO[str]*) – Stream to write into
- **home** (*str*) – `HOME` environment variable
- **path** (*str*) – `PATH` environment variable

Return type
None

cmkinitramfs.init.**do_init**(*out*)

Initialize the init environment

- Check the current PID is 1
- Mount `/proc`, `/sys`, `/dev`
- Set the kernel log level to 4 (`KERN_ERR` and higher priority)

Parameters
out (*IO[str]*) – Stream to write into

Return type
None

cmkinitramfs.init.**do_cmdline**(*out*)

Parse the kernel command line for known parameters

Note: the command line is parsed up to "–", arguments after this are passed through to the final init process.

Parsed parameters:

- `init=<path to init>`: Set the program to run as init process after the initramfs.
- `debug`: Enable debugging, see `rd.debug`.
- `quiet`: Enable quiet mode, see `rd.quiet`.
- `rd.break={init|rootfs|mount}`: Stops the boot process, defaults to `rootfs`. See *Breakpoint*.
- `rd.debug`: Enable debugging mode: output verbose informations. If quiet mode is disabled, enable shell trace (with `set -x`).
- `rd.panic`: On fatal error: cause a kernel panic rather than dropping into a shell.
- `rd.quiet`: Enable quiet mode: reduce verbosity.

Parameters
out (*IO[str]*) – Stream to write into

Return type
None

cmkinitramfs.init.**do_keymap**(*out*, *keymap_file*, *unicode=True*)

Load a keymap

Parameters

- **out** (*IO[str]*) – Stream to write into
- **keymap_file** (*str*) – Absolute path of the file to load
- **unicode** (*bool*) – Set the keyboard in unicode mode (rather than ASCII)

Return type
None

cmkinitramfs.init.**do_module**(*out*, *module*, **args*)

Load a kernel module

Parameters

- **out** (*IO[str]*) – Stream to write into

- **module** (*str*) – Name of the module to load
- **args** (*str*) – Arguments for the module (passed to `modprobe`)

Return type
None

cmkinitramfs.init.**do_break**(*out*, *breakpoint_*, *scripts=()*)

Drop into a shell if rd.break is set

Parameters

- **out** (*IO[str]*) – Stream to write into
- **breakpoint** – Which breakpoint to check
- **scripts** (*Iterable[str]*) – User commands to run before the breakpoint
- **breakpoint_** (Breakpoint) –

Return type
None

cmkinitramfs.init.**do_switch_root**(*out*, *newroot*, *init='/sbin/init'*)

Cleanup and switch root

- Print debugging information
- Restore kernel log level (set to boot-time default if not possible)
- Kill all processes
- Unmount `/dev`, `/sys`, `/proc`
- Switch root

Parameters

- **out** (*IO[str]*) – Stream to write into
- **newroot** (Data) – Data to use as new root
- **init** (*str*) – Init process to execute from the new root

Return type
None

cmkinitramfs.init.**mkinit**(*out*, *root*, *mounts=()*, *keymap=None*, *modules=None*, *scripts=None*)

Create the init script

Parameters

- **out** (*IO[str]*) – Stream to write into
- **root** (Data) – `Data` to use as rootfs
- **mounts** (*Iterable[Data]*) – `Data` needed in addition of rootfs
- **keymap** (*Optional[str]*) – Path of the keymap to load, `None` means no keymap
- **modules** (*Optional[Mapping[str, Iterable[str]]]*) – Kernel modules to be loaded in the initramfs: `{module: (arg, ...)}`. `module` is the module name string, and `(arg, ...)` is the iterable with the module parameters.

- **scripts** (*Optional[Mapping[Breakpoint, Iterable[str]]]*) – User commands to run. `{breakpoint: commands}`: `breakpoint` is the *Breakpoint* where the commands will be run. `commands` is the iterable with the commands.

Return type
None

6.2 data

Module providing the Data class for init script data sources management

The *Data* class defines an abstract object containing data, it has multiple subclasses for multiple types of data. The main methods of those classes are *Data.load()*, *Data.unload()*, and *Data.set_final()*.

Most functions will write into a stream (text file) the content of the init script. Use a `io.StringIO` if you need to use strings rather than a stream.

class cmkinitramfs.data.**Data**

Bases: `object`

Base class representing any data on the system

This is an abstract class representing data on the system. Its main methods are *load()* and *unload()*. *set_final()* declare the object as required for the final boot environment (e.g. root fs, usr fs), this will prevent the data from being unloaded.

Parameters

- **files** – Files directly needed in the initramfs. Each file is a tuple in the format (`src`, `dest`), where `src` is the source file on the current system, and `dest` is the destination in the initramfs (relative to its root directory). If `dest` is `None`, then `src` is used.
- **execs** – Executables directly needed in the initramfs. Same format as `files`.
- **libs** – Libraries directly needed in the initramfs. Same format as `files`.
- **busybox** – Busybox compatible commands needed in the initramfs. Any commands that are compatible with Busybox's implementation should be added. Exception: special shell built-in commands and reserved words are guaranteed to be available and *can* be ommitted (a list is defined in *cmkinitramfs.initramfs.SHELL_SPECIAL_BUILTIN* and *cmkinitramfs.initramfs.SHELL_RESERVED_WORDS*).
- **kmods** – Kernel modules directly needed in the initramfs. Each module is a tuple in the format `(module, params)`, where `params` is a tuple of module parameters (may be empty).
- **_need** – Loading and runtime dependencies
- **_lneed** – Loading only dependencies
- **_needed_by** – Reverse dependencies
- **_is_final** – The *Data* should not be unloaded
- **_is_loaded** – The *Data* is currently loaded

__str__()

Get the name of the data

This string may be quoted with simple quotes in the script. This **has** to be implemented by subclasses.

Return type
str

_post_load(*out*)

This function does post loading cleanup

If the object is a loading dependency only, it will load all its reverse dependencies in order to be unloaded as soon as possible. Unloading quickly can be useful when dealing with sensitive data (e.g. a LUKS key). It should be called from *load()* after the actual loading of the data.

Parameters
out (*IO[str]*) – Stream to write into

Return type
None

_post_unload(*out*)

This function does post unloading cleanup

It removes itself from the _needed_by reverse dependencies of all its dependencies, and check if the dependency can be unloaded. This method should be called from *unload()* after the actual unloading of the data. This *should not* be called if the data is not loaded.

Parameters
out (*IO[str]*) – Stream to write into

Return type
None

_pre_load(*out*)

This function does preparation for loading the Data

Loads all the needed dependencies. It should be called from *load()* before the actual loading of the data. This method *should not* be called if the *Data* is already loaded.

Parameters
out (*IO[str]*) – Stream to write into

Raises
DataError – Already loaded

Return type
None

_pre_unload(*out*)

This function does pre unloading sanity checks

It should be called from *unload()* before the actual unloading of the data.

Parameters
out (*IO[str]*) – Stream to write into

Raises
DataError – Not loaded or dependency issue

Return type
None

add_dep(*dep*)

Add a *Data* object to the hard dependencies

Parameters
dep (*Data*) –

Return type
None

`add_load_dep`(*dep*)

Add a *Data* object to the loading dependencies

Parameters
dep (Data) –

Return type
None

`classmethod initialize`(*out*)

Initialize the data class

Initialize the environment for the use of this data class: define needed functions and variables. A Data class should be initialized only once in the init script, before the first *Data.load()* call of the class.

Default initialization is a no-op and should be redefined by subclasses. Subclasses should call their parent *Data* class' *Data.initialize()* method.

Parameters
out (*IO[str]*) – Stream to write into

Return type
None

`is_final`()

Returns a bool indicating if the *Data* is final

Return type
bool

`iter_all_deps`()

Recursivelly get dependencies

Returns
Iterator over all the dependencies

Return type
Iterator[Data]

`load`(*out*)

This function loads the data

It should be redefined by subclasses, this definition is a no-op only dealing with dependencies.

Before loading, this function should load the dependencies with *_pre_load()*. After loading, this function should unload unnecessary dependencies with *_post_load()*. This method *should not* be called if the data is already loaded.

Parameters
out (*IO[str]*) – Stream to write into

Return type
None

`path`()

Get the path of this data

This function provides a string allowing access to data from within the init environment, this string can be a path or a command in a subshell (e.g. `"$(findfs UUID=foobar)"`). This string should be ready to be used in the script without being quoted nor escaped. This **has** to be implemented by subclasses.

Return type
str

set_final()

This function set the data object as final

This means the data is required by the final boot environment and should never be unloaded (it would be pointless). This will also mark its hard dependencies as final.

Return type
None

unload(*out*)

This function unloads data

It should be redefined by subclasses, this definition is a no-op only dealing with dependencies.

Before unloading, this function should check for any dependency error, with *_pre_unload()*. After unloading, this function should unload all unneeded dependencies, with *_post_unload()*.

Parameters
out (*IO[str]*) – Stream to write into

Return type
None

class cmkinitramfs.data.**PathData**(*datapath*)

Bases: *Data*

Absolute path

Parameters
datapath (*str*) – Path of the data

class cmkinitramfs.data.**UuidData**(*uuid*, *partition=False*)

Bases: *Data*

UUID of a data

The UUID can be a filesystem UUID, or other UUID known by other *Data* classes (e.g. a MD UUID).

Parameters

- **uuid** (*str*) – UUID of the data
- **partition** (*bool*) – If True, the UUID is treated as a partition UUID

class cmkinitramfs.data.**LabelData**(*label*, *partition=False*)

Bases: *Data*

Label of a data

The label can be a filesystem or partition label, or a label known by other *Data* classes.

Parameters

- **label** (*str*) – Label of the data
- **partition** (*bool*) – If True, the label is treated as a partition label

class cmkinitramfs.data.**LuksData**(*source*, *name*, *key=None*, *header=None*, *discard=False*)

Bases: *Data*

LUKS encrypted block device

Parameters

- **source** (Data) – *Data* to unlock (crypto_LUKS volume), it will be set as a hard dependency

- **name** (*str*) – Name for the LUKS volume
- **key** (*Optional[Data]*) – *Data* to use as key file, it will be set as a load dependency
- **header** (*Optional[Data]*) – *Data* containing the LUKS header, it will be set as a load dependency
- **discard** (*bool*) – Enable discards

class cmkinitramfs.data.**LvmData**(*vg_name*, *lv_name*)

Bases: *Data*

LVM logical volume

Parameters

- **vg_name** (*str*) – Name of the volume group
- **lv_name** (*str*) – Name of the logical volume

__lvm_conf()

Create LVM config in `/etc/lvm/lvmlocal.conf`

This override some configurations specific to the initramfs environment.

Note: if `/etc/lvm/lvmlocal.conf` exists, we append to it, which may cause duplicate configuration warnings from LVM.

Parameters
out (*IO[str]*) –

Return type
None

class cmkinitramfs.data.**MountData**(*source*, *mountpoint*, *filesystem*, *options='ro'*)

Bases: *Data*

Mount point

Parameters

- **source** (Data) – *Data* to use as source (e.g. /dev/sda1, my-luks-data), it will be set as a hard dependency
- **mountpoint** (*str*) – Absolute path of the mountpoint
- **filesystem** (*str*) – Filesystem (used for `mount -t filesystem`)
- **options** (*str*) – Mount options

__fun_fsck()

Define the mount_fsck function

This function takes any number of arguments, which will be passed to `fsck`. This function checks the return code of the `fsck` command and acts accordingly.

This functions calls fsck with `$@`. It checks the return code of `fsck` and :

- No error: returns 0.
- Non fatal error: prints an error and returns 0.
- Non fatal error requiring reboot: prints an error and reboot.
- Fatal error: returns 1.

Parameters
out (*IO[str]*) – Stream to write into

Return type
None

class cmkinitramfs.data.**MdData**(*sources*, *name*)

Bases: *Data*

MD RAID

Parameters

- **sources** (*Tuple[Data, ...]*) – *Data* to use as sources (e.g. /dev/sda1 and /dev/sdb1; or UUID=foo), they will be set as hard dependencies
- **name** (*str*) – Name for the MD device

Raises
ValueError – No *Data* source

class cmkinitramfs.data.**CloneData**(*source*, *dest*)

Bases: *Data*

Clone a *Data* to another

Parameters

- **source** (Data) – *Data* to use as source, it will be set as a load dependency
- **dest** (Data) – *Data* to use as destination, it will be set as a hard dependency

class cmkinitramfs.data.**ZFSPoolData**(*pool*, *cache*)

Bases: *Data*

ZFS pool

Parameters

- **pool** (*str*) – Pool name
- **cache** (*Optional[Data]*) – *Data* containing a ZFS cache file, it will be set as a load dependency

class cmkinitramfs.data.**ZFSCryptData**(*pool*, *dataset*, *key=None*)

Bases: *Data*

ZFS encrypted dataset

Parameters

- **pool** (ZFSPoolData) – *ZFSPoolData* containing the encrypted dataset, it will be set as a hard dependency
- **dataset** (*str*) – Dataset name
- **key** (*Optional[Data]*) – *Data* to use as key file, it will be set as a load dependency

exception cmkinitramfs.data.**DataError**

Bases: Exception

Error in the *Data* object

6.3 initramfs

Module providing functions and classes to build an initramfs

This library provides a class *Initramfs* which handles the content of an initramfs. This class supports creating an initramfs tree inside a directlry. It also can generate a CPIO file list compatible with the Linux kernel's `gen_init_cpio` utility. (See https://www.kernel.org/doc/html/latest/filesystems/ramfs-rootfs-initramfs.html for mor details.)

The main function is `mkinitramfs()` to build the complete initramfs.

cmkinitramfs.initramfs.**busybox_get_applets**(*busybox_exec*)

Get BusyBox applets

Parameters
busybox_exec (*str*) – BusyBox executable (e.g. `busybox`)

Returns
Iterator of absolute paths of BusyBox applets

Raises
subprocess.CalledProcessError – Error during `busybox_exec`

Return type
Iterator[str]

cmkinitramfs.initramfs.**mkcpio_from_dir**(*src*, *dest*)

Create CPIO archive from a given directory

Parameters

- **src** (*str*) – Directory from which the archive is created
- **dest** (*IO[bytes]*) – Destination stream of the CPIO data

Raises
subprocess.CalledProcessError – Error during `find` or `cpio`

Return type
None

cmkinitramfs.initramfs.**mkcpio_from_list**(*src*, *dest*)

Create CPIO archive from a given CPIO list

Parameters

- **src** (*str*) – Path of the CPIO list
- **dest** (*IO[bytes]*) – Destination stream of the CPIO data

Raises
subprocess.CalledProcessError – Error during `gen_init_cpio`

Return type
None

cmkinitramfs.initramfs.**keymap_build**(*src*, *dest*, *unicode=True*)

Generate a binary keymap from a keymap name

This keymap can then be loaded with `loadkmap` from the initramfs environment.

Parameters

- **src** (*str*) – Name of the keymap to convert, can be a keyboard layout name or a file path

- **dest** (*IO[bytes]*) – Destination stream to write into
- **unicode** (*bool*) – Generate a unicode keymap (rather than ASCII)

Raises
subprocess.CalledProcessError – Error during `loadkeys`

Return type
None

cmkinitramfs.initramfs.**SHELL_SPECIAL_BUILTIN = frozenset({'.', ':', 'break', 'continue', 'eval', 'exec', 'exit', 'export', 'readonly', 'return', 'set', 'shift', 'times', 'trap', 'unset'})**

Set of shell special built-in commands. They are guaranteed to be available in the initramfs' `/bin/sh`.

cmkinitramfs.initramfs.**SHELL_RESERVED_WORDS = frozenset({'!', 'case', 'do', 'done', 'elif', 'else', 'esac', 'fi', 'for', 'if', 'in', 'then', 'until', 'while', '{', '}'})**

Set of shell reserved words. They are guaranteed to be available in the initramfs' `/bin/sh`.

class cmkinitramfs.initramfs.**Initramfs**(*user=0*, *group=0*, *binroot='/'*, *kernels=None*)

Bases: `object`

An initramfs archive

Parameters

- **user** (*int*) – Default user to use when creating items
- **group** (*int*) – Default group to use when creating items
- **binroot** (*str*) – Root directory where binary files are found (executables and libraries)
- **kernels** (*Set[str]*) – Kernel versions of the initramfs, defaults to the running kernel version
- **items** – Items in the initramfs

__contains__(*path*)

Check if a path exists on the initramfs

Parameters
path (*str*) – Path to check

Returns
`True` if `path` exists on the initramfs, `False` otherwise

Return type
bool

__iter__()

Iterate over the `Item` instances in the initramfs

Return type
Iterator[Item]

__normalize()

Normalize a path for the initramfs filesystem

Strip /usr[/local] directory, warns if spaces are present in the path.

Parameters
path (*str*) – Destination path to normalize

Returns
Normalized path

Raises
ValueError – Invalid path

Return type
str

add_busybox(*needed=()*, *sys_busybox=None*)

Add busybox and its applets to the initramfs

Applets will be ignored if a file with the same path already exists. To avoid collision, this method should be called after all needed files have been added.

If any command listed in `needed` is not available in busybox (either as an applet, a shell special built-in, or a reserved word), the default system one will be included (from `PATH`).

Parameters

- **needed** (*Iterable[str]*) – Needed Busybox compatible shell commands
- **sys_busybox** (*Optional[str]*) – Busybox executable to use to get the list of applets, defaults to the one in `PATH`

Raises

- **FileNotFoundError** – Busybox executable or ELF dependency not found
- ***MergeError*** – Destination file exists and is different, or missing parent directory (raised from *add_item()*, not raised for applets)

Return type
None

add_executable(*src*, *dest=None*, *mode=None*)

Add an executable to the initramfs

Parameters

- **src** (*str*) – Path or base name of the executable to add, if it is not a path, it is searched on the system with `find_exec()`
- **dest** (*Optional[str]*) – Absolute path of the destination, relative to the initramfs root, defaults to the path of the source executable
- **mode** (*Optional[int]*) – File permissions to use, defaults to same as `src`

Raises

- **FileNotFoundError** – Executable or ELF dependency not found
- ***MergeError*** – Destination file exists and is different, or missing parent directory (raised from *add_item()*)

Return type
None

add_file(*src*, *dest=None*, *mode=None*)

Add a file to the initramfs

If the file is a symlink, it is dereferenced. If it is a dynamically linked ELF file, its dependencies are also added.

Parameters

- **src** (*str*) – Absolute or relative path of the source file
- **dest** (*Optional[str]*) – Absolute path of the destination, relative to the initramfs root, defaults to `src`
- **mode** (*Optional[int]*) – File permissions to use, defaults to same as `src`

Raises

- **FileNotFoundError** – Source file or ELF dependency not found
- ***MergeError*** – Destination file exists and is different, or missing parent directory (raised from *add_item()*)

Return type
None

add_item(*new_item*)

Add an item to the initramfs

If an identical item is already present, merges them together.

Parameters
new_item (Item) – `Item` instance to add

Raises
MergeError – Item cannot be merged into the initramfs (missing parent directory or file conflict)

Return type
None

add_kmod(*module*, *mode=None*)

Add a kernel module to the initramfs

Parameters

- **module** (*str*) – Path or name of the kernel module to add
- **mode** (*Optional[int]*) – File permissions to use, defaults to same as `module`

Return type
None

add_library(*src*, *dest=None*, *mode=None*)

Add a library to the initramfs

Parameters

- **src** (*str*) – Path or base name of the library to add, if it is not a path, it is searched on the system with `find_lib()`
- **dest** (*Optional[str]*) – Absolute path of the destination, relative to the initramfs root, defaults to the path of the source library
- **mode** (*Optional[int]*) – File permissions to use, defaults to same as `src`

Raises

- **FileNotFoundError** – Library or ELF dependency not found
- ***MergeError*** – Destination file exists and is different, or missing parent directory (raised from *add_item()*)

Return type
None

build_to_cpio_list(*dest*)

Write a CPIO list into a file

This list is compatible with Linux's `gen_init_cpio`. See `Item.build_to_cpio_list()`.

Parameters
dest (*IO[str]*) – Stream in which the list is written

Return type
None

build_to_directory(*dest*, *do_nodes=True*)

Copy or create all items to a real filesystem

See `Item.build_to_directory()`.

Parameters

- **dest** (*str*) – Path to use as root directory of the initramfs
- **do_nodes** (*bool*) – Also creates `Node` items, (used for debugging: `CAP_MKNOD` is needed to create some special devices)

Return type
None

mkdir(*path*, *mode=493*, *parents=False*)

Create a directory on the initramfs

Parameters

- **path** (*str*) – Absolute path of the directory, relative to the initramfs root
- **mode** (*int*) – File permissions to use
- **parents** (*bool*) – If `True`, missing parent directories will also be created

Raises
MergeError – Destination file exists and is different, or missing parent directory (raised from *add_item()*)

Return type
None

6.4 item

Module providing the Item class for files in the initramfs

Each file type of the initramfs has an *Item* subclass. Those classes provide methods used by `Initramfs` generation methods.

exception cmkinitramfs.item.**MergeError**

Bases: `Exception`

Cannot merge an Item into another

class cmkinitramfs.item.**Item**

Bases: `ABC`

An object within the initramfs

__contains__(*path*)

Check if this item is present at the given path in the initramfs

This method should be overriden by subclasses which add files to the initramfs.

Returns
`True` if `path` is part of this *Item*'s destination paths, `False` otherwise

Parameters
path (*str*) –

Return type
bool

__iter__()

Get the paths of this item within the initramfs

This method should be overriden by subclasses which add files to the initramfs.

Returns
Iterator over this *Item*'s destination paths

Return type
Iterator[str]

static build_from_cpio_list(*data*)

Build an Item from a string

This string should respect the format of `gen_init_cpio`.

Parameters
data (*str*) – String to parse

Returns
Item corresponding to `data`

Raises
ValueError – Invalid string

Return type
Item

abstract build_to_cpio_list()

String representing the item

The string is formatted to be compatible with the `gen_init_cpio` tool from the Linux kernel. This method has to be defined by subclasses.

Return type
str

abstract build_to_directory(*base_dir*)

Add this item to a real filesystem

This will copy or create a file on a real filesystem. This method has to be defined by subclasses.

Parameters
base_dir (*str*) – Path to use as root directory (e.g. using `/tmp/initramfs`, `/bin/ls` will be copied to `/tmp/initramfs/bin/ls`)

Return type
None

`is_mergeable`(*other*)

Check if two items can be merged together

By default, two items can only be merged if they are equal.

Parameters
other (`Item`) – *Item* to merge into `self`

Returns
`True` if the items can be merged, `False` otherwise

Return type
bool

`merge`(*other*)

Merge two items together

Default merge is just a no-op. Subclasses can override this as done by *File* to handle hardlink of identical files.

Parameters
other (`Item`) – *Item* to merge into `self`

Raises
MergeError – Cannot merge the items

Return type
None

`class cmkinitramfs.item.File`(*mode*, *user*, *group*, *dests*, *src*, *data_hash*, *chunk_size=65536*)

Bases: *Item*

Normal file within the initramfs

Parameters

- **mode** (*int*) – Permissions (e.g. 0o644)
- **user** (*int*) – Owner user (UID)
- **group** (*int*) – Owner group (GID)
- **dests** (*Set[str]*) – Paths in the initramfs (hard-linked)
- **src** (*str*) – Source file to copy (not unique to the file)
- **data_hash** (*bytes*) – Hash of the file (can be obtained with `hash_file()`)
- **chunk_size** (*int*) – Chunk size to use when copying the file

`class cmkinitramfs.item.Directory`(*mode*, *user*, *group*, *dest*)

Bases: *Item*

Directory within the initramfs

Parameters

- **mode** (*int*) – Permissions (e.g. 0o644)
- **user** (*int*) – Owner user (UID)
- **group** (*int*) – Owner group (GID)
- **dest** (*str*) – Path in the initramfs

class cmkinitramfs.item.**Node**(*mode*, *user*, *group*, *dest*, *nodetype*, *major*, *minor*)

Bases: *Item*

Special file within the initramfs

Parameters

- **mode** (*int*) – Permissions (e.g. 0o644)
- **user** (*int*) – Owner user (UID)
- **group** (*int*) – Owner group (GID)
- **dest** (*str*) – Path in the initramfs
- **nodetype** (*NodeType*) – Type of node (block, character)
- **major** (*int*) – Major number of the node
- **minor** (*int*) – Minor number of the node

class cmkinitramfs.item.**Symlink**(*mode*, *user*, *group*, *dest*, *target*)

Bases: *Item*

Symlinks within the initramfs

Parameters

- **mode** (*int*) – Permissions (e.g. 0o644)
- **user** (*int*) – Owner user (UID)
- **group** (*int*) – Owner group (GID)
- **dest** (*str*) – Path in the initramfs
- **target** (*str*) – Link target

class cmkinitramfs.item.**Pipe**(*mode*, *user*, *group*, *dest*)

Bases: *Item*

Named pipe (FIFO) within the initramfs

Parameters

- **mode** (*int*) – Permissions (e.g. 0o644)
- **user** (*int*) – Owner user (UID)
- **group** (*int*) – Owner group (GID)
- **dest** (*str*) – Path in the initramfs

class cmkinitramfs.item.**Socket**(*mode*, *user*, *group*, *dest*)

Bases: *Item*

Named socket within the initramfs

Parameters

- **mode** (*int*) – Permissions (e.g. 0o644)
- **user** (*int*) – Owner user (UID)
- **group** (*int*) – Owner group (GID)
- **dest** (*str*) – Path in the initramfs

6.5 bin

Module providing functions to manage binaries and executables

This library also provides utilities like *find_exec()*. Multiple helper functions are also available (e.g. *find_elf_deps_iter()* and *find_elf_deps_set()* to list libraries needed by an ELF executable).

class cmkinitramfs.bin.**ELFIncompatibleError**

Bases: ELFError

The ELF files are incompatible

cmkinitramfs.bin.**parse_ld_path**(*ld_path=None*, *origin=''*, *root='/'*)

Parse a colon-delimited list of paths and apply ldso rules

Note the special handling as dictated by the ldso:

- Empty paths are equivalent to $PWD
- $ORIGIN is expanded to the path of the given file, `origin`

Parameters

- **ld_path** (*Optional[str]*) – Colon-delimited string of paths, defaults to the LD_LIBRARY_PATH environment variable
- **origin** (*str*) – Directory containing the ELF file being parsed (used for $ORIGIN), defaults to an empty string
- **root** (*str*) – Path to prepend to all paths found

Returns
Iterator over the processed paths

Return type
Iterator[str]

cmkinitramfs.bin.**parse_ld_so_conf_iter**(*conf_path=None*, *root='/'*)

Parse a ldso config file

This should handle comments, whitespace, and "include" statements.

Parameters

- **conf_path** (*Optional[str]*) – Path of the ldso config file to parse, defaults to `{root}/etc/ld.so.conf`
- **root** (*str*) – Path to prepend to all paths found

Returns
Iterator over the processed paths

Return type
Iterator[str]

cmkinitramfs.bin.**parse_ld_so_conf_tuple**(*conf_path=None*, *root='/'*)

Parse a ldso config file

Cached version of *parse_ld_so_conf_iter()*, returning a tuple.

Parameters

- **conf_path** (*Optional[str]*) – Path of the ldso config file to parse, defaults to `{root}/etc/ld.so.conf`
- **root** (*str*) – Path to prepend to all paths found

Returns
Tuple with the processed paths

Return type
Tuple[str, ...]

cmkinitramfs.bin.**_get_default_libdirs**(*root='/'*)

Get the default library directories

Parameters
root (*str*) – Root directory to check for library directories

Returns
Libdirs in the initramfs

Return type
Tuple[str, ...]

cmkinitramfs.bin.**_get_libdir**(*arch*, *root='/'*)

Get the libdir corresponding to a binary class

Parameters

- **arch** (*int*) – Binary class (e.g. `32` or `64`)
- **root** (*str*) – Directory where libdirs are searched

Returns
Libdir in the initramfs

Return type
str

cmkinitramfs.bin.**_is_elf_compatible**(*elf1*, *elf2*)

See if two ELFs are compatible

This compares the aspects of the ELF to see if they're compatible: bit size, endianness, machine type, and operating system.

Parameters

- **elf1** (*ELFFile*) – First ELF object
- **elf2** (*ELFFile*) – Second ELF object

Returns
`True` if compatible, `False` otherwise

Return type
bool

cmkinitramfs.bin.**_get_elf_arch**(*elf1*, *elf2*)

Open elf2, check compatibility, and return ELF architecture

Parameters

- **elf1** (*Union[ELFFile, str]*) – First ELF
- **elf2** (*Union[ELFFile, str]*) – Second ELF

Returns
Architecture of the ELF files (32 or 64)

Raises
- **OSError** – Could not open an ELF file
- ***ELFIncompatibleError*** – ELFs are incompatible
- **ELFError** – File is not an ELF file

Return type
int

cmkinitramfs.bin.**_find_elf_deps_iter**(*elf*, *origin*, *root='/'*)

Iterates over the dependencies of an ELF file

Backend of *find_elf_deps_iter()*.

Parameters
- **elf** (*ELFFile*) – Elf file to parse
- **origin** (*str*) – Directory containing the ELF binary (real path as provided by `os.path.realpath()`, used for $ORIGIN)
- **root** (*str*) – Path to prepend to all paths found

Returns
Same as *find_elf_deps_iter()*

Raises
FileNotFoundError – Dependency not found

Return type
Iterator[*Tuple*[str, str]]

cmkinitramfs.bin.**find_elf_deps_iter**(*src*, *root='/'*)

Iterates over the dependencies of an ELF file

Read an ELF file to search dynamic library dependencies. For each dependency, find it on the system (using RPATH, LD_LIBRARY_PATH, RUNPATH, ld.so.conf, and default library directories).

If the library is in a path encoded in the ELF binary (RPATH or RUNPATH), `dep_dest = dep_src`, otherwise use a default library directory according to the type of binary (`/lib`, `/lib64`, `/lib32`).

If the file is not an ELF file, returns an empty iterator.

Parameters
- **src** (*str*) – File to find dependencies for
- **root** (*str*) – Path to prepend to all paths found

Returns
Iterator of `(dep_src, dep_dest)`, with `dep_src` the path of the dependency on the current system, and `dep_dest` the path of the dependency on the initramfs

Raises
FileNotFoundError – Dependency not found

Return type
Iterator[*Tuple*[str, str]]

cmkinitramfs.bin.**find_elf_deps_set**(*src*, *root='/'*)

Find dependencies of an ELF file

Cached version of *find_elf_deps_iter()*.

Parameters

- **src** (*str*) – File to find dependencies for
- **root** (*str*) – Path to prepend to all paths found

Returns
Set of `(dep_src, dep_dest)`, see *find_elf_deps_iter()*.

Raises
FileNotFoundError – Dependency not found

Return type
FrozenSet[*Tuple*[str, str]]

cmkinitramfs.bin.**find_lib_iter**(*lib*, *compat=None*, *root='/'*)

Search a library in the system, with globbing

Same as *find_lib()* but uses `glob.glob()` to find matching libraries.

Parameters

- **lib** (*str*) – Glob pattern for the library to search (e.g. `libgcc_s.*`)
- **compat** (*Optional[str]*) – Path to a binary that the library must be compatible with (checked with *_is_elf_compatible()*), defaults to `{root}/bin/sh`
- **root** (*str*) – Path to prepend to all paths found

Returns
Iterator over `(lib_src, lib_dest)`, see *find_lib()*

Raises
FileNotFoundError – Library not found

Return type
Iterator[*Tuple*[str, str]]

cmkinitramfs.bin.**find_lib**(*lib*, *compat=None*, *root='/'*)

Search a library in the system, without globbing

Uses `ld.so.conf` and `LD_LIBRARY_PATH`.

Libraries will be installed in the default library directory in the initramfs.

Parameters

- **lib** (*str*) – Library to search (e.g. `libgcc_s.so.1`)
- **compat** (*Optional[str]*) – Path to a binary that the library must be compatible with (checked with *_is_elf_compatible()*), defaults to `{root}/bin/sh`
- **root** (*str*) – Path to prepend to all paths found

Returns
`(lib_src, lib_dest)`, with `lib_src` the absolute path of the library on the current system, and `lib_dest` the absolute path of the library on the initramfs

Raises
FileNotFoundError – Library not found

Return type
Tuple[str, str]

cmkinitramfs.bin.**parse_path**(*path=None*, *root='/'*)

Parse PATH variable

Parameters

- **path** (*Optional[str]*) – PATH string to parse, default to the PATH environment variable
- **root** (*str*) – Path to prepend to all paths found

Returns
Iterator over the processed paths

Return type
Iterator[str]

cmkinitramfs.bin.**find_exec**(*executable*, *compat=None*, *root='/'*)

Search an executable in the system

Uses the PATH environment variable.

Parameters

- **executable** (*str*) – Executable to search
- **compat** (*Optional[str]*) – Path to a binary that the executable must be compatible with (checked with *_is_elf_compatible()*), defaults to {root}/bin/sh
- **root** (*str*) – Path to prepend to all paths found

Returns
(src_path, dest_path) with src_path the path of the executable on root, and dest_path the default path of the executable on the initramfs.

Returns
Absolute path of the executable

Raises
FileNotFoundError – Executable not found

Return type
Tuple[str, str]

cmkinitramfs.bin.**_get_all_kmods**(*kernel*)

Get all kernel modules on the system

Parameters
kernel (*str*) – Target kernel version

Returns
Set with the absolute path of the modules

Return type
FrozenSet[str]

cmkinitramfs.bin.**KMOD_DIR = '/lib/modules'**

Kernel modules will be searched in {KMOD_DIR}/{KERNEL}/**/*.ko

cmkinitramfs.bin.**find_kmod_deps**(*path*)

Get kernel module dependencies

Parameters
path (*str*) – Path of the kernel module to parse

Returns
Set with the dependencies' names

Raises
subprocess.CalledProcessError – Error during `modinfo`

Return type
FrozenSet[str]

`cmkinitramfs.bin.`**find_kmod**(*module*, *kernel*)

Search a kernel module on the system

Parameters

- **module** (*str*) – Name of the kernel module
- **kernel** (*str*) – Target kernel version

Returns
Absolute path of the kernel module on the system

Raises
FileNotFoundError – Kernel module not found

Return type
Optional[str]

6.6 entry

Entry point module for cmkinitramfs

class `cmkinitramfs.entry.`**Config**(*root*, *mounts*, *keymap*, *files*, *execs*, *libs*, *busybox*, *init_path*, *cmkcpiodir_opts*, *cmkcpiolist_opts*, *modules*, *has_modules_manual*, *scripts*)

Bases: `object`

Configuration informations

Parameters

- **root** (Data) – Rootfs data needed to boot
- **mounts** (*Iterable[Data]*) – Non-rootfs datas needed to boot
- **keymap** (*Optional[Tuple[str, str, str]]*) – Keymap information tuple `(source, build, dest)`: `source`: keymap to convert, `build`: converted keymap, `dest`: keymap path within the initramfs
- **files** (*Iterable[Tuple[str, Optional[str]]]*) – User configured files, see `cmkinitramfs.init.Data.files`
- **execs** (*Iterable[Tuple[str, Optional[str]]]*) – User configured executables, see `cmkinitramfs.init.Data.files`
- **libs** (*Iterable[Tuple[str, Optional[str]]]*) – User configured libraries, see `cmkinitramfs.init.Data.files`
- **busybox** (*Iterable[str]*) – Needed executables compatibles with busybox implementation

- **init_path** (*str*) – Path where the init script will be generated
- **cmkcpiodir_opts** (*str*) – Default options for cmkcpiodir
- **cmkcpiolist_opts** (*str*) – Default options for cmkcpiolist
- **modules** (*Mapping[str, Iterable[str]]*) – Kernel modules to be loaded in the initramfs: `{module: (arg, ...)}`. See *cmkinitramfs.init.mkinit()*.
- **scripts** (*Mapping[Breakpoint, Iterable[str]]*) – User scripts to run at given breakpoints. See `scripts` for *cmkinitramfs.init.mkinit()*.
- **has_modules_manual** (*bool*) –

cmkinitramfs.entry.**read_config**(*config_file=None*)

Read a configuration file and generate data structures from it

Parameters
config_file (*Optional[str]*) – Configuration file to use. Defaults to, in order: `CMKINITCFG` environment variable, `./cmkinitramfs.ini`, `/etc/cmkinitramfs.ini`.

Returns
Configuration dictionnary, described by *Config*

Raises
ValueError – Config file parsing error

Return type
Config

cmkinitramfs.entry.**entry_cmkinit**()

Main entry point of the module

Return type
None

cmkinitramfs.entry.**entry_findlib**()

Entry point for the findlib utility

Return type
None

cmkinitramfs.entry.**entry_cmkcpiolist**()

Entry point for cmkcpiolist

Return type
None

cmkinitramfs.entry.**entry_cmkcpiodir**()

Entry point for cmkcpiodir

Return type
None

6.7 utils

Module providing miscellaneous utilities used by cmkinitramfs

cmkinitramfs.utils.**removeprefix**(*string*, *prefix*)

Remove a prefix from a string

Add support for `str.removeprefix()` for Python < 3.9.

Parameters

- **string** (*str*) – String to remove prefix from
- **prefix** (*str*) – Prefix to remove

Return type
str

cmkinitramfs.utils.**normpath**(*path*)

Normalize path (actually eliminates double slashes)

Parameters
path (*str*) – Path to normalize

Return type
str

cmkinitramfs.utils.**hash_file**(*filepath*, *chunk_size=65536*)

Calculate the SHA512 of a file

Parameters

- **filepath** (*str*) – Path of the file to hash
- **chunk_size** (*int*) – Number of bytes per chunk of file to hash

Returns
File hash in a `bytes` object

Return type
bytes

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX

Symbols

A

B

C

D

E

F

H

I

K

L

M

N

P

R

S

U

Z