Rocky Enterprise Linux 9.2 Manual Pages on command 'loop-control.4'

C:\>man loop-control.4

LOOP(4) Linux Programmer's Manual LOOP(4)

NAME

loop, loop-control - loop devices

SYNOPSIS

#include <linux/loop.h>

DESCRIPTION

The loop device is a block device that maps its data blocks not to a physical de? vice such as a hard disk or optical disk drive, but to the blocks of a regular file in a filesystem or to another block device. This can be useful for example to pro? vide a block device for a filesystem image stored in a file, so that it can be mounted with the mount(8) command. You could do

```
$ dd if=/dev/zero of=file.img bs=1MiB count=10
$ sudo losetup /dev/loop4 file.img
$ sudo mkfs -t ext4 /dev/loop4
$ sudo mkdir /myloopdev
$ sudo mount /dev/loop4 /myloopdev
```

See losetup(8) for another example.

A transfer function can be specified for each loop device for encryption and de? cryption purposes.

The following ioctl(2) operations are provided by the loop block device:

LOOP_SET_FD

Associate the loop device with the open file whose file descriptor is passed

as the (third) ioctl(2) argument.

LOOP_CLR_FD

Disassociate the loop device from any file descriptor.

LOOP_SET_STATUS

Set the status of the loop device using the (third) ioctl(2) argument. This argument is a pointer to loop_info structure, defined in <linux/loop.h> as:

```
struct loop_info {
    int         lo_number;          /* ioctl r/o */
    dev_t       lo_device;          /* ioctl r/o */
    unsigned long lo_inode;         /* ioctl r/o */
    dev_t       lo_rdevice;         /* ioctl r/o */
    int         lo_offset;
    int         lo_encrypt_type;
    int         lo_encrypt_key_size;  /* ioctl w/o */
    int         lo_flags;           /* ioctl r/o */
    char        lo_name[LO_NAME_SIZE];
    unsigned char lo_encrypt_key[LO_KEY_SIZE];
                                    /* ioctl w/o */
    unsigned long lo_init[2];
    char        reserved[4];
};
```

The encryption type (lo_encrypt_type) should be one of LO_CRYPT_NONE, LO_CRYPT_XOR, LO_CRYPT_DES, LO_CRYPT_FISH2, LO_CRYPT_BLOW, LO_CRYPT_CAST128, LO_CRYPT_IDEA, LO_CRYPT_DUMMY, LO_CRYPT_SKIPJACK, or (since Linux 2.6.0) LO_CRYPT_CRYPTOAPI.

The lo_flags field is a bit mask that can include zero or more of the fol? lowing:

LO_FLAGS_READ_ONLY

The loopback device is read-only.

LO_FLAGS_AUTOCLEAR (since Linux 2.6.25)

The loopback device will autodestruct on last close.

LO_FLAGS_PARTSCAN (since Linux 3.2)

Allow automatic partition scanning.

LOOP_GET_STATUS

Get the status of the loop device. The (third) ioctl(2) argument must be a pointer to a struct loop_info.

LOOP_CHANGE_FD (since Linux 2.6.5)

Switch the backing store of the loop device to the new file identified file descriptor specified in the (third) ioctl(2) argument, which is an integer. This operation is possible only if the loop device is read-only and the new backing store is the same size and type as the old backing store.

LOOP_SET_CAPACITY (since Linux 2.6.30)

Resize a live loop device. One can change the size of the underlying back? ing store and then use this operation so that the loop driver learns about the new size. This operation takes no argument.

LOOP_SET_DIRECT_IO (since Linux 4.10)

Set DIRECT I/O mode on the loop device, so that it can be used to open back? ing file. The (third) ioctl(2) argument is an unsigned long value. A non-zero represents direct I/O mode.

LOOP_SET_BLOCK_SIZE (since Linux 4.14)

Set the block size of the loop device. The (third) ioctl(2) argument is an unsigned long value. This value must be a power of two in the range [512,pagesize]; otherwise, an EINVAL error results.

Since Linux 2.6, there are two new ioctl(2) operations:

LOOP_SET_STATUS64, LOOP_GET_STATUS64

These are similar to LOOP_SET_STATUS and LOOP_GET_STATUS described above but use the loop_info64 structure, which has some additional fields and a larger range for some other fields:

```
struct loop_info64 {
    uint64_t lo_device;                 /* ioctl r/o */
    uint64_t lo_inode;                  /* ioctl r/o */
    uint64_t lo_rdevice;                /* ioctl r/o */
    uint64_t lo_offset;
    uint64_t lo_sizelimit;/* bytes, 0 == max available */
    uint32_t lo_number;                 /* ioctl r/o */
    uint32_t lo_encrypt_type;
```

```
    uint32_t lo_encrypt_key_size;       /* ioctl w/o */
    uint32_t lo_flags;                  /* ioctl r/o */
    uint8_t  lo_file_name[LO_NAME_SIZE];
    uint8_t  lo_crypt_name[LO_NAME_SIZE];
    uint8_t  lo_encrypt_key[LO_KEY_SIZE]; /* ioctl w/o */
    uint64_t lo_init[2];
};
```

/dev/loop-control

Since Linux 3.1, the kernel provides the /dev/loop-control device, which permits an application to dynamically find a free device, and to add and remove loop devices from the system. To perform these operations, one first opens /dev/loop-control and then employs one of the following ioctl(2) operations:

LOOP_CTL_GET_FREE

Allocate or find a free loop device for use. On success, the device number is returned as the result of the call. This operation takes no argument.

LOOP_CTL_ADD

Add the new loop device whose device number is specified as a long integer in the third ioctl(2) argument. On success, the device index is returned as the result of the call. If the device is already allocated, the call fails with the error EEXIST.

LOOP_CTL_REMOVE

Remove the loop device whose device number is specified as a long integer in the third ioctl(2) argument. On success, the device number is returned as the result of the call. If the device is in use, the call fails with the error EBUSY.

FILES

/dev/loop*

The loop block special device files.

EXAMPLE

The program below uses the /dev/loop-control device to find a free loop device, opens the loop device, opens a file to be used as the underlying storage for the device, and then associates the loop device with the backing store. The following shell session demonstrates the use of the program:

```
$ dd if=/dev/zero of=file.img bs=1MiB count=10
10+0 records in
10+0 records out
10485760 bytes (10 MB) copied, 0.00609385 s, 1.7 GB/s
$ sudo ./mnt_loop file.img
loopname = /dev/loop5
```

Program source

```
#include <fcntl.h>
#include <linux/loop.h>
#include <sys/ioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define errExit(msg)    do { perror(msg); exit(EXIT_FAILURE); \
                        } while (0)
int
main(int argc, char *argv[])
{
    int loopctlfd, loopfd, backingfile;
    long devnr;
    char loopname[4096];
    if (argc != 2) {
        fprintf(stderr, "Usage: %s backing-file\n", argv[0]);
        exit(EXIT_FAILURE);
    }
    loopctlfd = open("/dev/loop-control", O_RDWR);
    if (loopctlfd == -1)
        errExit("open: /dev/loop-control");
    devnr = ioctl(loopctlfd, LOOP_CTL_GET_FREE);
    if (devnr == -1)
        errExit("ioctl-LOOP_CTL_GET_FREE");
    sprintf(loopname, "/dev/loop%ld", devnr);
    printf("loopname = %s\n", loopname);
```

```
        loopfd = open(loopname, O_RDWR);
        if (loopfd == -1)
            errExit("open: loopname");
        backingfile = open(argv[1], O_RDWR);
        if (backingfile == -1)
            errExit("open: backing-file");
        if (ioctl(loopfd, LOOP_SET_FD, backingfile) == -1)
            errExit("ioctl-LOOP_SET_FD");
        exit(EXIT_SUCCESS);
    }
```

SEE ALSO

losetup(8), mount(8)

COLOPHON

This page is part of release 5.05 of the Linux man-pages project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at https://www.kernel.org/doc/man-pages/.