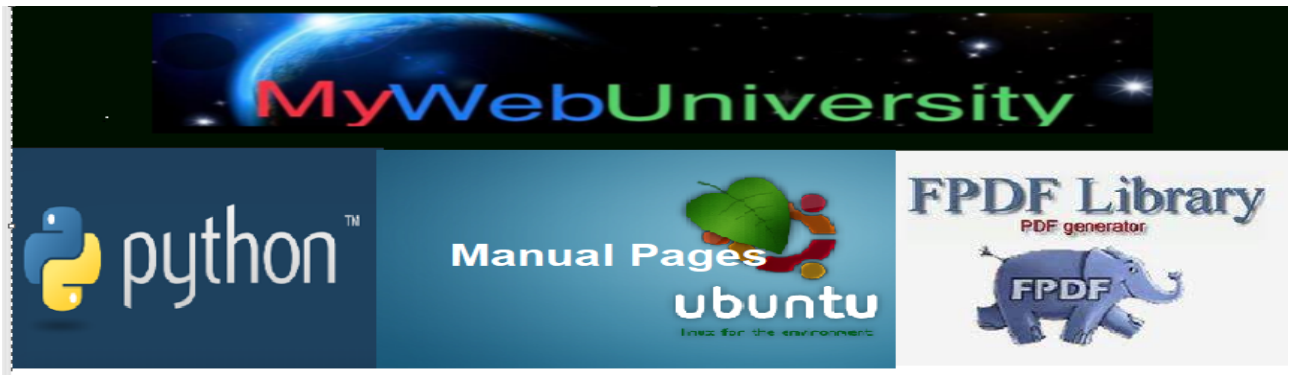

Full credit is given to the above companies including the OS that this PDF file was generated!

Linux Ubuntu 26.04 Manual Pages on command 'move_mount.2'

$ man move_mount.2

move_mount(2) System Calls Manual move_mount(2)

NAME

move_mount - move or attach mount object to filesystem

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

```
#include <fcntl.h>          /* Definition of AT_* constants */
#include <sys/mount.h>
int move_mount(int from_dirfd, const char *from_path,
               int to_dirfd, const char *to_path,
               unsigned int flags);
```

DESCRIPTION

The move_mount() system call is part of the suite of file-descriptor-based mount facilities in Linux.

move_mount() moves the mount object indicated by from_dirfd and from_path to the path indicated by to_dirfd and to_path. The mount object being moved can be an existing mount point in the current mount namespace, or a detached mount object created by fsmount(2) or open_tree(2) with OPEN_TREE_CLONE.

To access the source mount object or the destination mount point, no permissions are required on the object itself, but if either pathname is supplied, execute (search) permission is required on all of the directories specified in from_path or to_path.

The calling process must have the CAP_SYS_ADMIN capability in order to move or attach a mount object.

As with "*at()" system calls, move_mount() uses the from_dirfd and to_dirfd arguments in conjunction with the from_path and to_path arguments to determine the source and destination objects to operate on (respectively), as follows:

? If the pathname given in *_path is absolute, then the corresponding *_dirfd is ignored.

? If the pathname given in *_path is relative and the corresponding *_dirfd is the special value AT_FDCWD, then *_path is interpreted relative to the current working directory of the calling process (like open(2)).

? If the pathname given in *_path is relative, then it is interpreted relative to the di? rectory referred to by the corresponding file descriptor *_dirfd (rather than relative to the current working directory of the calling process, as is done by open(2) for a rela? tive pathname). In this case, the corresponding *_dirfd must be a directory that was opened for reading (O_RDONLY) or using the O_PATH flag.

? If *_path is an empty string, and flags contains the appropriate MOVE_MOUNT_*_EMPTY_PATH flag, then the corresponding file descriptor *_dirfd is operated on directly. In this case, the corresponding *_dirfd may refer to any type of file, not just a directory.

See openat(2) for an explanation of why the *_dirfd arguments are useful.

flags can be used to control aspects of the path lookup for both the source and destination objects, as well as other properties of the mount operation. A value for flags is con? structed by bitwise ORing zero or more of the following constants:

MOVE_MOUNT_F_EMPTY_PATH

If from_path is an empty string, operate on the file referred to by from_dirfd (which may have been obtained from open(2), fsmount(2), or open_tree(2)). In this case, from_dirfd may refer to any type of file, not just a directory. If from_dirfd is AT_FDCWD, move_mount() will operate on the current working di? rectory of the calling process.

This is the most common mechanism used to attach detached mount objects pro? duced by fsmount(2) and open_tree(2) to a mount point.

MOVE_MOUNT_T_EMPTY_PATH

As with MOVE_MOUNT_F_EMPTY_PATH, except operating on to_dirfd and to_path.

MOVE_MOUNT_F_SYMLINKS

If from_path references a symbolic link, then dereference it. The default be? havior for move_mount() is to not follow symbolic links.

MOVE_MOUNT_T_SYMLINKS

As with MOVE_MOUNT_F_SYMLINKS, except operating on to_dirfd and to_path.

MOVE_MOUNT_F_NO_AUTOMOUNT

Do not automount the terminal ("basename") component of from_path if it is a directory that is an automount point. This allows a mount object that has an automount point at its root to be moved and prevents unintended triggering of an automount point. This flag has no effect if the automount point has al? ready been mounted over.

MOVE_MOUNT_T_NO_AUTOMOUNT

As with MOVE_MOUNT_F_NO_AUTOMOUNT, except operating on to_dirfd and to_path. This allows an automount point to be manually mounted over.

MOVE_MOUNT_SET_GROUP (since Linux 5.15)

Add the attached private-propagation mount object indicated by to_dirfd and to_path into the mount propagation "peer group" of the attached non-private-propagation mount object indicated by from_dirfd and from_path.

Unlike other move_mount() operations, this operation does not move or attach any mount objects. Instead, it only updates the metadata of attached mount objects. (Also, take careful note of the argument order?the mount object be? ing modified by this operation is the one specified by to_dirfd and to_path.)

This makes it possible to first create a mount tree consisting only of private mounts and then configure the desired propagation layout afterwards. (See the "SHARED SUBTREES" section of mount_namespaces(7) for more information about mount propagation and peer groups.)

MOVE_MOUNT_BENEATH (since Linux 6.5)

If the path indicated by to_dirfd and to_path is an existing mount object, rather than attaching or moving the mount object indicated by from_dirfd and from_path on top of the mount stack, attach or move it beneath the current top mount on the mount stack.

After using MOVE_MOUNT_BENEATH, it is possible to umount(2) the top mount in order to reveal the mount object which was attached beneath it earlier. This allows for the seamless (and atomic) replacement of intricate mount trees, which can further be used to "upgrade" a mount tree with a newer version.

This operation has several restrictions:

? Mount objects cannot be attached beneath the filesystem root, including

cases where the filesystem root was configured by chroot(2) or pivot_root(2). To mount beneath the filesystem root, pivot_root(2) must be used.

- The target path indicated by to_dirfd and to_path must not be a detached mount object, such as those produced by open_tree(2) with OPEN_TREE_CLONE or fsmount(2).
- The current top mount of the target path's mount stack and its parent mount must be in the calling process's mount namespace.
- The caller must have sufficient privileges to unmount the top mount of the target path's mount stack, to prove they have privileges to reveal the underlying mount.
- Mount propagation events triggered by this move_mount() operation (as described in mount_namespaces(7)) are calculated based on the parent mount of the current top mount of the target path's mount stack.
- The target path's mount cannot be an ancestor in the mount tree of the source mount object.
- The source mount object must not have any overmounts, otherwise it would be possible to create "shadow mounts" (i.e., two mounts mounted on the same parent mount at the same mount point).
- It is not possible to move a mount beneath a top mount if the parent mount of the current top mount propagates to the top mount itself. Otherwise, MOVE_MOUNT_BENEATH would cause the mount object to be propagated to the top mount from the parent mount, defeating the purpose of using MOVE_MOUNT_BENEATH.
- It is not possible to move a mount beneath a top mount if the parent mount of the current top mount propagates to the mount object being mounted beneath. Otherwise, this would cause a similar propagation issue to the previous point, also defeating the purpose of using MOVE_MOUNT_BENEATH.

If from_dirfd is a mount object file descriptor and move_mount() is operating on it directly, from_dirfd will remain associated with the mount object after move_mount() succeeds, so you may repeatedly use from_dirfd with move_mount(2) and/or "*at()" system calls as many times as necessary.

RETURN VALUE

On success, move_mount() returns 0. On error, -1 is returned, and errno is set to indicate the error.

ERRORS

EACCES Search permission is denied for one of the directories in the path prefix of one of from_path or to_path. (See also path_resolution(7).)

EBADF One of from_dirfd or to_dirfd is not a valid file descriptor.

EFAULT One of from_path or to_path is NULL or a pointer to a location outside the calling process's accessible address space.

EINVAL Invalid flag specified in flags.

EINVAL The path indicated by from_dirfd and from_path is not a mount object.

EINVAL The mount object type of the source mount object and target inode are not compatible (i.e., the source is a file but the target is a directory, or vice-versa).

EINVAL The source mount object or target path are not in the calling process's mount name? space (or an anonymous mount namespace of the calling process).

EINVAL The source mount object's parent mount has shared mount propagation, and thus cannot be moved (as described in mount_namespaces(7)).

EINVAL The source mount has MS_UNBINDABLE child mounts but the target path resides on a mount tree with shared mount propagation, which would otherwise cause the unbindable mounts to be propagated (as described in mount_namespaces(7)).

EINVAL MOVE_MOUNT_BENEATH was attempted, but one of the listed restrictions was violated.

ELOOP Too many symbolic links encountered when resolving one of from_path or to_path.

ENAMETOOLONG

One of from_path or to_path is longer than PATH_MAX.

ENOENT A component of one of from_path or to_path does not exist.

ENOENT One of from_path or to_path is an empty string, but the corresponding MOVE_MOUNT_*_EMPTY_PATH flag is not specified in flags.

ENOTDIR

A component of the path prefix of one of from_path or to_path is not a directory, or one of from_path or to_path is relative and the corresponding from_dirfd or to_dirfd is a file descriptor referring to a file other than a directory.

ENOMEM The kernel could not allocate sufficient memory to complete the operation.

EPERM The calling process does not have the required CAP_SYS_ADMIN capability.

STANDARDS

Linux.

HISTORY

Linux 5.2, glibc 2.36.

EXAMPLES

move_mount() can be used to move attached mounts like the following:

```
move_mount(AT_FDCWD, "/a", AT_FDCWD, "/b", 0);
```

This would move the mount object mounted on /a to /b. The above procedure is functionally equivalent to the following mount operation using mount(2):

```
mount("/a", "/b", NULL, MS_MOVE, NULL);
```

move_mount() can also be used in conjunction with file descriptors returned from open_tree(2) or open(2):

```
int fd = open_tree(AT_FDCWD, "/mnt", 0); /* open("/mnt", O_PATH); */
move_mount(fd, "", AT_FDCWD, "/mnt2", MOVE_MOUNT_F_EMPTY_PATH);
move_mount(fd, "", AT_FDCWD, "/mnt3", MOVE_MOUNT_F_EMPTY_PATH);
move_mount(fd, "", AT_FDCWD, "/mnt4", MOVE_MOUNT_F_EMPTY_PATH);
```

This would move the mount object mounted at /mnt to /mnt2, then /mnt3, and then /mnt4.

If the source mount object indicated by from_dirfd and from_path is a detached mount object, move_mount() can be used to attach it to a mount point:

```
int fsfd, mntfd;
fsfd = fsopen("ext4", FSOPEN_CLOEXEC);
fsconfig(fsfd, FSCONFIG_SET_STRING, "source", "/dev/sda1", 0);
fsconfig(fsfd, FSCONFIG_SET_FLAG, "user_xattr", NULL, 0);
fsconfig(fsfd, FSCONFIG_CMD_CREATE, NULL, NULL, 0);
mntfd = fsmount(fsfd, FSMOUNT_CLOEXEC, MOUNT_ATTR_NODEV);
move_mount(mntfd, "", AT_FDCWD, "/home", MOVE_MOUNT_F_EMPTY_PATH);
```

This would create a new filesystem configuration context for ext4, configure it, create a detached mount object, and then attach it to /home. The above procedure is functionally equivalent to the following mount operation using mount(2):

```
mount("/dev/sda1", "/home", "ext4", MS_NODEV, "user_xattr");
```

The same operation also works with detached bind-mounts created with open_tree(2) with OPEN_TREE_CLONE:

```
int mntfd = open_tree(AT_FDCWD, "/home/cyphar", OPEN_TREE_CLONE);
move_mount(mntfd, "", AT_FDCWD, "/root", MOVE_MOUNT_F_EMPTY_PATH);
```

This would create a new bind-mount of /home/cyphar as a detached mount object, and then at? tach it to /root. The above procedure is functionally equivalent to the following mount op? eration using mount(2):

```
mount("/home/cyphar", "/root", NULL, MS_BIND, NULL);
```

SEE ALSO

fsconfig(2), fsmount(2), fsopen(2), fspick(2), mount(2), mount_setattr(2), open_tree(2), mount_namespaces(7)