# mount(2) — Linux manual page

NAME | SYNOPSIS | DESCRIPTION | RETURN VALUE | ERRORS | VERSIONS | CONFORMING TO | NOTES | SEE ALSO | COLOPHON



## NAME top

mount - mount filesystem

## SYNOPSIS top

```
#include <sys/mount.h>

int mount(const char *source, const char *target,
          const char *filesystemtype, unsigned long mountflags,
          const void *data);
```

## DESCRIPTION top

**mount**() attaches the filesystem specified by *source* (which is often a pathname referring to a device, but can also be the pathname of a directory or file, or a dummy string) to the location (a directory or file) specified by the pathname in *target*.

Appropriate privilege (Linux: the **CAP_SYS_ADMIN** capability) is required to mount filesystems.

Values for the *filesystemtype* argument supported by the kernel are listed in */proc/filesystems* (e.g., "btrfs", "ext4", "jfs", "xfs", "vfat", "fuse", "tmpfs", "cgroup", "proc", "mqueue", "nfs", "cifs", "iso9660"). Further types may become available when the appropriate modules are loaded.

The *data* argument is interpreted by the different filesystems. Typically it is a string of comma-separated options understood by this filesystem. See mount(8) for details of the options available for each filesystem type. This argument may be specified as NULL, if there are no options.

A call to **mount**() performs one of a number of general types of operation, depending on the bits specified in *mountflags*. The choice of which operation to perform is determined by testing the bits set in *mountflags*, with the tests being conducted in the order listed here:

* Remount an existing mount: *mountflags* includes **MS_REMOUNT**.

* Create a bind mount: *mountflags* includes **MS_BIND**.

* Change the propagation type of an existing mount: *mountflags* includes one of **MS_SHARED**, **MS_PRIVATE**, **MS_SLAVE**, or **MS_UNBINDABLE**.

* Move an existing mount to a new location: *mountflags* includes **MS_MOVE**.

* Create a new mount: *mountflags* includes none of the above flags.

Each of these operations is detailed later in this page. Further flags may be specified in *mountflags* to modify the behavior of **mount**(), as described below.

### Additional mount flags

The list below describes the additional flags that can be specified in *mountflags*. Note that some operation types ignore some or all of these flags, as described later in this page.

**MS_NOEXEC**
: Do not allow programs to be executed from this filesystem.

**MS_RDONLY**
: Mount filesystem read-only.

**MS_REC** (since Linux 2.4.11)
: Used in conjunction with **MS_BIND** to create a recursive bind mount, and in conjunction with the propagation type flags to recursively change the propagation type of all of the mounts in a subtree. See below for further details.

### Creating a bind mount

If *mountflags* includes **MS_BIND** (available since Linux 2.4), then perform a bind mount. A bind mount makes a file or a directory subtree visible at another point within the single directory hierarchy. Bind mounts may cross filesystem boundaries and span chroot(2) jails.

The *filesystemtype* and *data* arguments are ignored.

The remaining bits (other than **MS_REC**, described below) in the *mountflags* argument are also ignored. (The bind mount has the same mount options as the underlying mount.) However, see the discussion of remounting above, for a method of making an existing bind mount read-only.

By default, when a directory is bind mounted, only that directory is mounted; if there are any submounts under the directory tree, they are not bind mounted. If the **MS_REC** flag is also specified, then a recursive bind mount operation is performed: all submounts under the *source* subtree (other than unbindable mounts) are also bind mounted at the corresponding location in the *target* subtree.

### Changing the propagation type of an existing mount

If *mountflags* includes one of **MS_SHARED**, **MS_PRIVATE**, **MS_SLAVE**, or **MS_UNBINDABLE** (all available since Linux 2.6.15), then the propagation type of an existing mount is changed. If more than one of these flags is specified, an error results.

The only other flags that can be specified while changing the propagation type are **MS_REC** (described below) and **MS_SILENT**

(which is ignored).

The *source*, *filesystemtype*, and *data* arguments are ignored.

The meanings of the propagation type flags are as follows:

**MS_SHARED**
: Make this mount shared. Mount and unmount events immediately under this mount will propagate to the other mounts that are members of this mount's peer group. Propagation here means that the same mount or unmount will automatically occur under all of the other mounts in the peer group. Conversely, mount and unmount events that take place under peer mounts will propagate to this mount.

**MS_PRIVATE**
: Make this mount private. Mount and unmount events do not propagate into or out of this mount.

**MS_SLAVE**
: If this is a shared mount that is a member of a peer group that contains other members, convert it to a slave mount. If this is a shared mount that is a member of a peer group that contains no other members, convert it to a private mount. Otherwise, the propagation type of the mount is left unchanged.

  When a mount is a slave, mount and unmount events propagate into this mount from the (master) shared peer group of which it was formerly a member. Mount and unmount events under this mount do not propagate to any peer.

  A mount can be the slave of another peer group while at the same time sharing mount and unmount events with a peer group of which it is a member.

**MS_UNBINDABLE**
: Make this mount unbindable. This is like a private mount, and in addition this mount can't be bind mounted. When a recursive bind mount (**mount**() with the **MS_BIND** and **MS_REC** flags) is performed on a directory subtree, any unbindable mounts within the subtree are automatically pruned (i.e., not replicated) when replicating that subtree to produce the target subtree.

By default, changing the propagation type affects only the *target* mount. If the **MS_REC** flag is also specified in *mountflags*, then the propagation type of all mounts under *target* is also changed.

For further details regarding mount propagation types (including the default propagation type assigned to new mounts), see mount_namespaces(7).

**Creating a new mount**

If none of **MS_REMOUNT**, **MS_BIND**, **MS_MOVE**, **MS_SHARED**, **MS_PRIVATE**, **MS_SLAVE**, or **MS_UNBINDABLE** is specified in *mountflags*, then **mount**() performs its default action: creating a new mount. *source* specifies the source for the new mount, and *target* specifies the directory at which to create the mount point.

The *filesystemtype* and *data* arguments are employed, and further bits may be specified in *mountflags* to modify the behavior of the

call.

## RETURN VALUE top

On success, zero is returned. On error, -1 is returned, and *errno* is set to indicate the error.

## VERSIONS top

The definitions of **MS_DIRSYNC**, **MS_MOVE**, **MS_PRIVATE**, **MS_REC**, **MS_RELATIME**, **MS_SHARED**, **MS_SLAVE**, **MS_STRICTATIME**, and **MS_UNBINDABLE** were added to glibc headers in version 2.12.

## CONFORMING TO top

This function is Linux-specific and should not be used in programs intended to be portable.

## NOTES top

Since Linux 2.4 a single filesystem can be mounted at multiple mount points, and multiple mounts can be stacked on the same mount point.

**Parental relationship between mounts**

Each mount has a parent mount. The overall parental relationship of all mounts defines the single directory hierarchy seen by the processes within a mount namespace.

The parent of a new mount is defined when the mount is created. In the usual case, the parent of a new mount is the mount of the filesystem containing the directory or file at which the new mount is attached. In the case where a new mount is stacked on top of an existing mount, the parent of the new mount is the previous mount that was stacked at that location.

The parental relationship between mounts can be discovered via the */proc/[pid]/mountinfo* file (see below).

**/proc/[pid]/mounts and /proc/[pid]/mountinfo**

The Linux-specific */proc/[pid]/mounts* file exposes the list of mounts in the mount namespace of the process with the specified ID. The */proc/[pid]/mountinfo* file exposes even more information about mounts, including the propagation type and mount ID information that makes it possible to discover the parental relationship between mounts. See proc(5) and mount_namespaces(7) for details of this file.

## SEE ALSO top

mountpoint(1), chroot(2), ioctl_iflags(2), **mount_settatr**(2), pivot_root(2), umount(2), mount_namespaces(7), path_resolution(7), findmnt(8), lsblk(8), mount(8), umount(8)

## COLOPHON top

This page is part of release 5.13 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/.