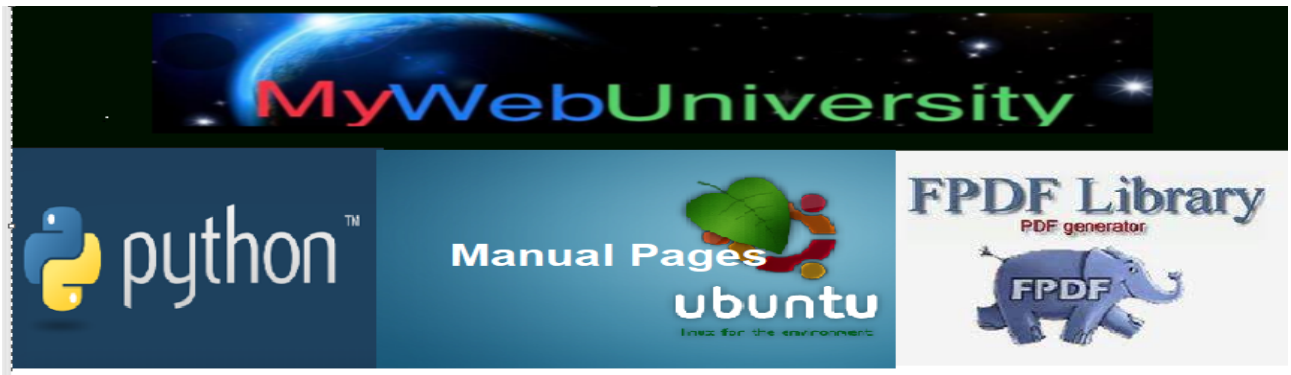

Full credit is given to the above companies including the OS that this PDF file was generated!

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

$ man fsconfig.2

fsconfig(2) System Calls Manual fsconfig(2)

NAME

fsconfig - configure new or existing filesystem context

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

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

int fsconfig(int fd, unsigned int cmd,
             const char *_Nullable key,
             const void *_Nullable value, int aux);
```

DESCRIPTION

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

fsconfig() is used to supply parameters to and issue commands against the filesystem config? uration context associated with the file descriptor fd. Filesystem configuration contexts can be created with fsopen(2) or be instantiated from an extant filesystem instance with fspick(2).

The cmd argument indicates the command to be issued. Some commands supply parameters to the context (equivalent to mount options specified with mount(8)), while others are meta-opera? tions on the filesystem context. The list of valid cmd values are:

FSCONFIG_SET_FLAG

Set the flag parameter named by key. value must be NULL, and aux must be 0.

FSCONFIG_SET_STRING

Set the string parameter named by key to the value specified by value. value points to a null-terminated string, and aux must be 0.

FSCONFIG_SET_BINARY

Set the blob parameter named by key to the contents of the binary blob speci? fied by value. value points to the start of a buffer that is aux bytes in length.

FSCONFIG_SET_FD

Set the file parameter named by key to the open file description referenced by the file descriptor aux. value must be NULL.

You may also use FSCONFIG_SET_STRING for file parameters, with value set to a null-terminated string containing a base-10 representation of the file de? scriptor number. This mechanism is primarily intended for compatibility with older mount(2)-based programs, and only works for parameters that only accept file descriptor arguments.

FSCONFIG_SET_PATH

Set the path parameter named by key to the object at a provided path, resolved in a similar manner to openat(2). value points to a null-terminated pathname string, and aux is equivalent to the dirfd argument to openat(2). See ope? nat(2) for an explanation of the need for FSCONFIG_SET_PATH.

You may also use FSCONFIG_SET_STRING for path parameters, the behavior of which is equivalent to FSCONFIG_SET_PATH with aux set to AT_FDCWD.

FSCONFIG_SET_PATH_EMPTY

As with FSCONFIG_SET_PATH, except that if value is an empty string, the file descriptor specified by aux is operated on directly and may be any type of file (not just a directory). This is equivalent to the behavior of AT_EMPTY_PATH with most "*at()" system calls. If aux is AT_FDCWD, the parame? ter will be set to the current working directory of the calling process.

FSCONFIG_CMD_CREATE

This command instructs the filesystem driver to instantiate an instance of the filesystem in the kernel with the parameters specified in the filesystem con? figuration context. key and value must be NULL, and aux must be 0.

This command can only be issued once in the lifetime of a filesystem context.

If the operation succeeds, the filesystem context associated with file de?

scriptor fd now references the created filesystem instance, and is placed into a special "awaiting-mount" mode that allows you to use fsmount(2) to create a mount object from the filesystem instance. If the operation fails, in most cases the filesystem context is placed in a failed mode and cannot be used for any further fsconfig() operations (though you may still retrieve diagnostic messages through the message retrieval interface, as described in the corre? sponding subsection of fsopen(2)).

This command can only be issued against filesystem configuration contexts that were created with fsopen(2). In order to create a filesystem instance, the calling process must have the CAP_SYS_ADMIN capability.

An important thing to be aware of is that the Linux kernel will silently reuse extant filesystem instances depending on the filesystem type and the config? ured parameters (each filesystem driver has its own policy for how filesystem instances are reused). This means that the filesystem instance "created" by FSCONFIG_CMD_CREATE may, in fact, be a reference to an extant filesystem in? stance in the kernel. (For reference, this behavior also applies to mount(2).)

One side-effect of this behavior is that if an extant filesystem instance is reused, all parameters configured for this filesystem configuration context are silently ignored (with the exception of the ro and rw flag parameters; if the state of the read-only flag in the extant filesystem instance and the filesystem configuration context do not match, this operation will return EBUSY). This also means that FSCONFIG_CMD_RECONFIGURE commands issued against the "created" filesystem instance will also affect any mount objects associ? ated with the extant filesystem instance.

Programs that need to ensure that they create a new filesystem instance with specific parameters (notably, security-related parameters such as acl to en? able POSIX ACLs?as described in acl(5)) should use FSCONFIG_CMD_CREATE_EXCL instead.

FSCONFIG_CMD_CREATE_EXCL (since Linux 6.6)

As with FSCONFIG_CMD_CREATE, except that the kernel is instructed to not reuse extant filesystem instances. If the operation would be forced to reuse an ex? tant filesystem instance, this operation will return EBUSY instead.

As a result (unlike FSCONFIG_CMD_CREATE), if this operation succeeds then the calling process can be sure that all of the parameters successfully configured with fsconfig() will actually be applied to the created filesystem instance.

FSCONFIG_CMD_RECONFIGURE

This command instructs the filesystem driver to apply the parameters specified in the filesystem configuration context to the extant filesystem instance referenced by the filesystem configuration context. key and value must be NULL, and aux must be 0.

This is primarily intended for use with fspick(2), but may also be used to modify the parameters of a filesystem instance after FSCONFIG_CMD_CREATE was used to create it and a mount object was created using fsmount(2). In order to reconfigure an extant filesystem instance, the calling process must have the CAP_SYS_ADMIN capability.

If the operation succeeds, the filesystem context is reset but remains in reconfiguration mode and thus can be reused for subsequent FSCONFIG_CMD_RECONFIGURE commands. If the operation fails, in most cases the filesystem context is placed in a failed mode and cannot be used for any further fsconfig() operations (though you may still retrieve diagnostic messages through the message retrieval interface, as described in the corresponding subsection of fsopen(2)).

Parameters specified with FSCONFIG_SET_* do not take effect until a corresponding FSCONFIG_CMD_CREATE or FSCONFIG_CMD_RECONFIGURE command is issued.

RETURN VALUE

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

ERRORS

If an error occurs, the filesystem driver may provide additional information about the error through the message retrieval interface for filesystem configuration contexts. This additional information can be retrieved at any time by calling read(2) on the filesystem instance or filesystem configuration context referenced by the file descriptor fd. (See the "Message retrieval interface" subsection in fsopen(2) for more details on the message format.)

Even after an error occurs, the filesystem configuration context is not invalidated, and

thus can still be used with other fsconfig() commands. This means that users can probe sup?
port for filesystem parameters on a per-parameter basis, and adjust which parameters they
wish to set.

The error values given below result from filesystem type independent errors. Each filesys?
tem type may have its own special errors and its own special behavior. See the Linux kernel
source code for details.

EACCES A component of a path provided as a path parameter was not searchable. (See also
path_resolution(7).)

EACCES FSCONFIG_CMD_CREATE was attempted for a read-only filesystem without specifying the
'ro' flag parameter.

EACCES A specified block device parameter is located on a filesystem mounted with the
MS_NODEV option.

EBADF The file descriptor given by fd (or possibly by aux, depending on the command) is in?
valid.

EBUSY The filesystem context associated with fd is in the wrong state for the given com?
mand.

EBUSY The filesystem instance cannot be reconfigured as read-only with
FSCONFIG_CMD_RECONFIGURE because some programs still hold files open for writing.

EBUSY A new filesystem instance was requested with FSCONFIG_CMD_CREATE_EXCL but a matching
superblock already existed.

EFAULT One of the pointer arguments points to a location outside the calling process's ac?
cessible address space.

EINVAL fd does not refer to a filesystem configuration context or filesystem instance.

EINVAL One of the values of key, value, and/or aux were set to a non-zero value when cmd re?
quired that they be zero (or NULL).

EINVAL The parameter named by key cannot be set using the type specified with cmd.

EINVAL One of the source parameters referred to an invalid superblock.

ELOOP Too many links encountered during pathname resolution of a path argument.

ENAMETOOLONG
A path argument was longer than PATH_MAX.

ENOENT A path argument had a non-existent component.

ENOENT A path argument is an empty string, but cmd is not FSCONFIG_SET_PATH_EMPTY.

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

ENOTBLK

The parameter named by key must be a block device, but the provided parameter value was not a block device.

ENOTDIR

A component of the path prefix of a path argument was not a directory.

EOPNOTSUPP

The command given by cmd is not valid.

ENXIO The major number of a block device parameter is out of range.

EPERM The command given by cmd was FSCONFIG_CMD_CREATE, FSCONFIG_CMD_CREATE_EXCL, or FSCONFIG_CMD_RECONFIGURE, but the calling process does not have the required CAP_SYS_ADMIN capability.

STANDARDS

Linux.

HISTORY

Linux 5.2. glibc 2.36.

NOTES

Generic filesystem parameters

Each filesystem driver is responsible for parsing most parameters specified with fsconfig(), meaning that individual filesystems may have very different behavior when encountering parameters with the same name. In general, you should not assume that the behavior of fsconfig() when specifying a parameter to one filesystem type will match the behavior of the same parameter with a different filesystem type.

However, the following generic parameters apply to all filesystems and have unified behavior. They are set using the listed FSCONFIG_SET_* command.

ro and rw (FSCONFIG_SET_FLAG)

Configure whether the filesystem instance is read-only.

dirsync (FSCONFIG_SET_FLAG)

Make directory changes on this filesystem instance synchronous.

sync and async (FSCONFIG_SET_FLAG)

Configure whether writes on this filesystem instance will be made synchronous (as though the O_SYNC flag to open(2) was specified for all file opens in this filesystem instance).

lazytime and nolazytime (FSCONFIG_SET_FLAG)

Configure whether to reduce on-disk updates of inode timestamps on this filesystem instance (as described in the MS_LAZYTIME section of mount(2)).

mand and nomand (FSCONFIG_SET_FLAG)

Configure whether the filesystem instance should permit mandatory locking. Since Linux 5.15, mandatory locking has been deprecated and setting this flag is a no-op.

source (FSCONFIG_SET_STRING)

This parameter is equivalent to the source parameter passed to mount(2) for the same filesystem type, and is usually the pathname of a block device containing the filesystem. This parameter may only be set once per filesystem configuration context transaction.

In addition, any filesystem parameters associated with Linux Security Modules (LSMs) are also generic with respect to the underlying filesystem. See the documentation for the LSM you wish to configure for more details.

Mount attributes and filesystem parameters

Some filesystem parameters (traditionally associated with mount(8)-style options) have a sibling mount attribute with superficially similar user-facing behavior.

For a description of the distinction between mount attributes and filesystem parameters, see the "Mount attributes and filesystem parameters" subsection of mount_setattr(2).

CAVEATS

Filesystem parameter types

As a result of each filesystem driver being responsible for parsing most parameters speci? fied with fsconfig(), some filesystem drivers may have unintuitive behavior with regards to which FSCONFIG_SET_* commands are permitted to configure a given parameter.

In order for filesystem parameters to be backwards compatible with mount(2), they must be parseable as strings; this almost universally means that FSCONFIG_SET_STRING can also be used to configure them. However, other FSCONFIG_SET_* commands need to be opted into by each filesystem driver's parameter parser.

One of the most user-visible instances of this inconsistency is that many filesystems do not support configuring path parameters with FSCONFIG_SET_PATH (despite the name), which can lead to somewhat confusing EINVAL errors. (For example, the generic source parameter?which is usually a path?can only be configured with FSCONFIG_SET_STRING.)

When writing programs that use fsconfig() to configure parameters with commands other than FSCONFIG_SET_STRING, users should verify that the FSCONFIG_SET_* commands used to configure

each parameter are supported by the corresponding filesystem driver.

EXAMPLES

To illustrate the different kinds of flags that can be configured with fsconfig(), here are a few examples of some different filesystems being created:

```
int fsfd, mntfd;
fsfd = fsopen("tmpfs", FSOPEN_CLOEXEC);
fsconfig(fsfd, FSCONFIG_SET_FLAG, "inode64", NULL, 0);
fsconfig(fsfd, FSCONFIG_SET_STRING, "uid", "1234", 0);
fsconfig(fsfd, FSCONFIG_SET_STRING, "huge", "never", 0);
fsconfig(fsfd, FSCONFIG_SET_FLAG, "casefold", NULL, 0);
fsconfig(fsfd, FSCONFIG_CMD_CREATE, NULL, NULL, 0);
mntfd = fsmount(fsfd, FSMOUNT_CLOEXEC, MOUNT_ATTR_NOEXEC);
move_mount(mntfd, "", AT_FDCWD, "/tmp", MOVE_MOUNT_F_EMPTY_PATH);
fsfd = fsopen("erofs", FSOPEN_CLOEXEC);
fsconfig(fsfd, FSCONFIG_SET_STRING, "source", "/dev/loop0", 0);
fsconfig(fsfd, FSCONFIG_SET_FLAG, "acl", NULL, 0);
fsconfig(fsfd, FSCONFIG_SET_FLAG, "user_xattr", NULL, 0);
fsconfig(fsfd, FSCONFIG_CMD_CREATE_EXCL, NULL, NULL, 0);
mntfd = fsmount(fsfd, FSMOUNT_CLOEXEC, MOUNT_ATTR_NOSUID);
move_mount(mntfd, "", AT_FDCWD, "/mnt", MOVE_MOUNT_F_EMPTY_PATH);
```

Usually, specifying the same parameter named by key multiple times with fsconfig() causes the parameter value to be replaced. However, some filesystems may have unique behavior:

```
int fsfd, mntfd;
int lowerdirfd = open("/o/ctr/lower1", O_DIRECTORY | O_CLOEXEC);
fsfd = fsopen("overlay", FSOPEN_CLOEXEC);
/* "lowerdir+" appends to the lower dir stack each time */
fsconfig(fsfd, FSCONFIG_SET_FD, "lowerdir+", NULL, lowerdirfd);
fsconfig(fsfd, FSCONFIG_SET_STRING, "lowerdir+", "/o/ctr/lower2", 0);
fsconfig(fsfd, FSCONFIG_SET_STRING, "lowerdir+", "/o/ctr/lower3", 0);
fsconfig(fsfd, FSCONFIG_SET_STRING, "lowerdir+", "/o/ctr/lower4", 0);
fsconfig(fsfd, FSCONFIG_SET_STRING, "xino", "auto", 0);
fsconfig(fsfd, FSCONFIG_SET_STRING, "nfs_export", "off", 0);
fsconfig(fsfd, FSCONFIG_CMD_CREATE, NULL, NULL, 0);
```

```
mntfd = fsmount(fsfd, FSMOUNT_CLOEXEC, 0);
move_mount(mntfd, "", AT_FDCWD, "/mnt", MOVE_MOUNT_F_EMPTY_PATH);
```

And here is an example of how fspick(2) can be used with fsconfig() to reconfigure the para? meters of an extant filesystem instance attached to /proc:

```
int fsfd = fspick(AT_FDCWD, "/proc", FSPICK_CLOEXEC);
fsconfig(fsfd, FSCONFIG_SET_STRING, "hidepid", "ptraceable", 0);
fsconfig(fsfd, FSCONFIG_SET_STRING, "subset", "pid", 0);
fsconfig(fsfd, FSCONFIG_CMD_RECONFIGURE, NULL, NULL, 0);
```

SEE ALSO

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