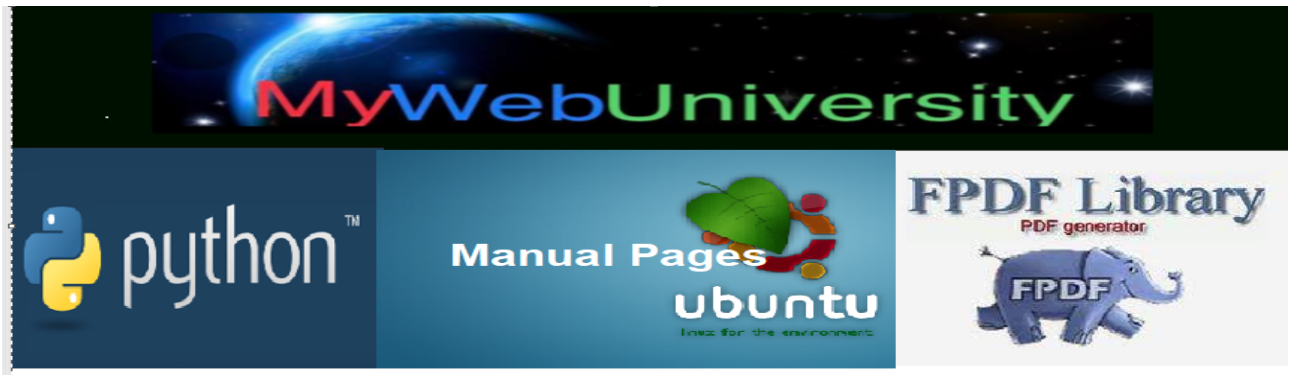

Full credit is given to the above companies including the OS that this PDF file was generated!

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

$ man statx.2

statx(2) System Calls Manual statx(2)

NAME

statx - get file status (extended)

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

```
#define _GNU_SOURCE         /* See feature_test_macros(7) */
#include <fcntl.h>          /* Definition of AT_* constants */
#include <sys/stat.h>
int statx(int dirfd, const char *_Nullable restrict path,
          int flags, unsigned int mask,
          struct statx *restrict statxbuf);
```

DESCRIPTION

This function returns information about a file, storing it in the buffer pointed to by statxbuf. The returned buffer is a structure of the following type:

```
   struct statx {
       __u32 stx_mask;        /* Mask of bits indicating
                                 filled fields */
       __u32 stx_blksize;     /* Block size for filesystem I/O */
       __u64 stx_attributes;  /* Extra file attribute indicators */
       __u32 stx_nlink;       /* Number of hard links */
       __u32 stx_uid;         /* User ID of owner */
       __u32 stx_gid;         /* Group ID of owner */
```

```
__u16 stx_mode;        /* File type and mode */
__u64 stx_ino;         /* Inode number */
__u64 stx_size;        /* Total size in bytes */
__u64 stx_blocks;      /* Number of 512B blocks allocated */
__u64 stx_attributes_mask;
                    /* Mask to show what's supported
                       in stx_attributes */
/* The following fields are file timestamps */
struct statx_timestamp stx_atime;  /* Last access */
struct statx_timestamp stx_btime;  /* Creation */
struct statx_timestamp stx_ctime;  /* Last status change */
struct statx_timestamp stx_mtime;  /* Last modification */
/* If this file represents a device, then the next two
   fields contain the ID of the device */
__u32 stx_rdev_major;  /* Major ID */
__u32 stx_rdev_minor;  /* Minor ID */
/* The next two fields contain the ID of the device
   containing the filesystem where the file resides */
__u32 stx_dev_major;   /* Major ID */
__u32 stx_dev_minor;   /* Minor ID */
__u64 stx_mnt_id;      /* Mount ID */
/* Direct I/O alignment restrictions */
__u32 stx_dio_mem_align;
__u32 stx_dio_offset_align;
__u64 stx_subvol;      /* Subvolume identifier */
/* Direct I/O atomic write limits */
__u32 stx_atomic_write_unit_min;
__u32 stx_atomic_write_unit_max;
__u32 stx_atomic_write_segments_max;
/* File offset alignment for direct I/O reads */
__u32 stx_dio_read_offset_align;
/* Direct I/O atomic write max opt limit */
__u32 stx_atomic_write_unit_max_opt;
```

```
};
```

The file timestamps are structures of the following type:

```
struct statx_timestamp {
    __s64 tv_sec;    /* Seconds since the Epoch (UNIX time) */
    __u32 tv_nsec;   /* Nanoseconds since tv_sec */
};
```

(Note that reserved space and padding is omitted.)

Invoking statx():

To access a file's status, no permissions are required on the file itself, but in the case of statx() with a pathname, execute (search) permission is required on all of the directo? ries in path that lead to the file.

statx() uses path, dirfd, and flags to identify the target file in one of the following ways:

An absolute pathname

If path begins with a slash, then it is an absolute pathname that identifies the tar? get file. In this case, dirfd is ignored.

A relative pathname

If path is a string that begins with a character other than a slash and dirfd is AT_FDCWD, then path is a relative pathname that is interpreted relative to the process's current working directory.

A directory-relative pathname

If path is a string that begins with a character other than a slash and dirfd is a file descriptor that refers to a directory, then path is a relative pathname that is interpreted relative to the directory referred to by dirfd. (See openat(2) for an explanation of why this is useful.)

By file descriptor

If path is an empty string (or NULL since Linux 6.11) and the AT_EMPTY_PATH flag is specified in flags (see below), then the target file is the one referred to by the file descriptor dirfd.

flags can be used to influence a pathname-based lookup. A value for flags is constructed by ORing together zero or more of the following constants:

AT_EMPTY_PATH

If path is an empty string (or NULL since Linux 6.11), operate on the file referred

to by dirfd (which may have been obtained using the open(2) O_PATH flag). In this case, dirfd can refer to any type of file, not just a directory.

If dirfd is AT_FDCWD, the call operates on the current working directory.

AT_NO_AUTOMOUNT

Don't automount the terminal ("basename") component of path if it is a directory that is an automount point. This allows the caller to gather attributes of an automount point (rather than the location it would mount). This flag has no effect if the mount point has already been mounted over.

The AT_NO_AUTOMOUNT flag can be used in tools that scan directories to prevent mass-automounting of a directory of automount points.

All of stat(2), lstat(2), and fstatat(2) act as though AT_NO_AUTOMOUNT was set.

AT_SYMLINK_NOFOLLOW

If path is a symbolic link, do not dereference it: instead return information about the link itself, like lstat(2).

flags can also be used to control what sort of synchronization the kernel will do when querying a file on a remote filesystem. This is done by ORing in one of the following val? ues:

AT_STATX_SYNC_AS_STAT

Do whatever stat(2) does. This is the default and is very much filesystem-specific.

AT_STATX_FORCE_SYNC

Force the attributes to be synchronized with the server. This may require that a network filesystem perform a data writeback to get the timestamps correct.

AT_STATX_DONT_SYNC

Don't synchronize anything, but rather just take whatever the system has cached if possible. This may mean that the information returned is approximate, but, on a net? work filesystem, it may not involve a round trip to the server - even if no lease is held.

The mask argument to statx() is used to tell the kernel which fields the caller is inter? ested in. mask is an ORed combination of the following constants:

STATX_TYPE	Want stx_mode & S_IFMT
STATX_MODE	Want stx_mode & ~S_IFMT
STATX_NLINK	Want stx_nlink
STATX_UID	Want stx_uid

STATX_GID Want stx_gid

STATX_ATIME Want stx_atime

STATX_MTIME Want stx_mtime

STATX_CTIME Want stx_ctime

STATX_INO Want stx_ino

STATX_SIZE Want stx_size

STATX_BLOCKS Want stx_blocks

STATX_BASIC_STATS [All of the above]

STATX_BTIME Want stx_btime

STATX_ALL The same as STATX_BASIC_STATS | STATX_BTIME.
It is deprecated and should not be used.

STATX_MNT_ID Want stx_mnt_id (since Linux 5.8)

STATX_DIOALIGN Want stx_dio_mem_align and stx_dio_offset_align.
(since Linux 6.1; support varies by filesystem)

STATX_MNT_ID_UNIQUE Want unique stx_mnt_id (since Linux 6.8)

STATX_SUBVOL Want stx_subvol
(since Linux 6.10; support varies by filesystem)

STATX_WRITE_ATOMIC Want stx_atomic_write_unit_min,
stx_atomic_write_unit_max,
stx_atomic_write_segments_max,
and stx_atomic_write_unit_max_opt.
(since Linux 6.11; support varies by filesystem)

STATX_DIO_READ_ALIGN Want stx_dio_read_offset_align.
(since Linux 6.14; support varies by filesystem)

Note that, in general, the kernel does not reject values in mask other than the above. (For an exception, see EINVAL in errors.) Instead, it simply informs the caller which values are supported by this kernel and filesystem via the statx.stx_mask field. Therefore, do not simply set mask to UINT_MAX (all bits set), as one or more bits may, in the future, be used to specify an extension to the buffer.

The returned information

The status information for the target file is returned in the statx structure pointed to by statxbuf. Included in this is stx_mask which indicates what other information has been re? turned. stx_mask has the same format as the mask argument and bits are set in it to indi?

cate which fields have been filled in.

It should be noted that the kernel may return fields that weren't requested and may fail to return fields that were requested, depending on what the backing filesystem supports. (Fields that are given values despite being unrequested can just be ignored.) In either case, stx_mask will not be equal mask.

If a filesystem does not support a field or if it has an unrepresentable value (for in? stance, a file with an exotic type), then the mask bit corresponding to that field will be cleared in stx_mask even if the user asked for it and a dummy value will be filled in for compatibility purposes if one is available (e.g., a dummy UID and GID may be specified to mount under some circumstances).

A filesystem may also fill in fields that the caller didn't ask for if it has values for them available and the information is available at no extra cost. If this happens, the cor? responding bits will be set in stx_mask.

Note: for performance and simplicity reasons, different fields in the statx structure may contain state information from different moments during the execution of the system call. For example, if stx_mode or stx_uid is changed by another process by calling chmod(2) or chown(2), stat() might return the old stx_mode together with the new stx_uid, or the old stx_uid together with the new stx_mode.

Apart from stx_mask (which is described above), the fields in the statx structure are:

stx_blksize

The "preferred" block size for efficient filesystem I/O. (Writing to a file in smaller chunks may cause an inefficient read-modify-rewrite.)

stx_attributes

Further status information about the file (see below for more information).

stx_nlink

The number of hard links on a file.

stx_uid

This field contains the user ID of the owner of the file.

stx_gid

This field contains the ID of the group owner of the file.

stx_mode

The file type and mode. See inode(7) for details.

stx_ino

The inode number of the file.

stx_size

The size of the file (if it is a regular file or a symbolic link) in bytes. The size of a symbolic link is the length of the pathname it contains, without a terminating null byte.

stx_blocks

The number of blocks allocated to the file on the medium, in 512-byte units. (This may be smaller than stx_size/512 when the file has holes.)

stx_attributes_mask

A mask indicating which bits in stx_attributes are supported by the VFS and the filesystem.

stx_atime

The file's last access timestamp.

stx_btime

The file's creation timestamp.

stx_ctime

The file's last status change timestamp.

stx_mtime

The file's last modification timestamp.

stx_dev_major

stx_dev_minor

The device on which this file (inode) resides.

stx_rdev_major

stx_rdev_minor

The device that this file (inode) represents if the file is of block or character de? vice type.

stx_mnt_id

If using STATX_MNT_ID, this is the mount ID of the mount containing the file. This is the same number reported by name_to_handle_at(2) and corresponds to the number in the first field in one of the records in /proc/self/mountinfo.

If using STATX_MNT_ID_UNIQUE, this is the unique mount ID of the mount containing the file. This is the number reported by listmount(2) and is the ID used to query the mount with statmount(2). It is guaranteed to not be reused while the system is run?

ning.

stx_dio_mem_align

The alignment (in bytes) required for user memory buffers for direct I/O (O_DIRECT) on this file, or 0 if direct I/O is not supported on this file.

STATX_DIOALIGN (stx_dio_mem_align and stx_dio_offset_align) is supported on block de? vices since Linux 6.1. The support on regular files varies by filesystem; it is sup? ported by ext4, f2fs, and xfs since Linux 6.1.

stx_dio_offset_align

The alignment (in bytes) required for file offsets and I/O segment lengths for direct I/O (O_DIRECT) on this file, or 0 if direct I/O is not supported on this file. This will only be nonzero if stx_dio_mem_align is nonzero, and vice versa.

stx_dio_read_offset_align

The alignment (in bytes) required for file offsets and I/O segment lengths for direct I/O reads (O_DIRECT) on this file. If zero, the limit in stx_dio_offset_align ap? plies for reads as well. If non-zero, this value must be smaller than or equal to stx_dio_offset_align which must be provided by the file system if requested by the application. The memory alignment in stx_dio_mem_align is not affected by this value.

STATX_DIO_READ_ALIGN (stx_dio_offset_align) is supported by xfs on regular files since Linux 6.14.

stx_subvol

Subvolume number of the current file.

Subvolumes are fancy directories, i.e., they form a tree structure that may be walked recursively. Support varies by filesystem; it is supported by bcachefs and btrfs since Linux 6.10.

stx_atomic_write_unit_min

stx_atomic_write_unit_max

The minimum and maximum sizes (in bytes) supported for direct I/O (O_DIRECT) on the file to be written with torn-write protection. These values are each guaranteed to be a power-of-2.

STATX_WRITE_ATOMIC (stx_atomic_write_unit_min, stx_atomic_write_unit_max, and stx_atomic_write_segments_max) is supported on block devices since Linux 6.11. The support on regular files varies by filesystem; it is supported by xfs and ext4 since

Linux 6.13.

stx_atomic_write_unit_max_opt

The maximum size (in bytes) which is optimised for writes issued with torn-write protection. If non-zero, this value will not exceed the value in stx_atomic_write_unit_max and will not be less than the value in stx_atomic_write_unit_min. A value of zero indicates that stx_atomic_write_unit_max is the optimised limit. Slower writes may be experienced when the size of the write exceeds stx_atomic_write_unit_max_opt (when non-zero).

stx_atomic_write_segments_max

The maximum number of elements in an array of vectors for a write with torn-write protection enabled. See RWF_ATOMIC flag for pwritev2(2).

For further information on the above fields, see inode(7).

File attributes

The stx_attributes field contains a set of ORed flags that indicate additional attributes of the file. Note that any attribute that is not indicated as supported by stx_attributes_mask has no usable value here. The bits in stx_attributes_mask correspond bit-by-bit to stx_attributes.

The flags are as follows:

STATX_ATTR_COMPRESSED

The file is compressed by the filesystem and may take extra resources to access.

STATX_ATTR_IMMUTABLE

The file cannot be modified: it cannot be deleted or renamed, no hard links can be created to this file and no data can be written to it. See chattr(1).

STATX_ATTR_APPEND

The file can only be opened in append mode for writing. Random access writing is not permitted. See chattr(1).

STATX_ATTR_NODUMP

File is not a candidate for backup when a backup program such as dump(8) is run. See chattr(1).

STATX_ATTR_ENCRYPTED

A key is required for the file to be encrypted by the filesystem.

STATX_ATTR_VERITY (since Linux 5.5)

The file has fs-verity enabled. It cannot be written to, and all reads from it will

be verified against a cryptographic hash that covers the entire file (e.g., via a Merkle tree).

STATX_ATTR_WRITE_ATOMIC (since Linux 6.11)

The file supports torn-write protection.

STATX_ATTR_DAX (since Linux 5.8)

The file is in the DAX (cpu direct access) state. DAX state attempts to minimize software cache effects for both I/O and memory mappings of this file. It requires a file system which has been configured to support DAX.

DAX generally assumes all accesses are via CPU load / store instructions which can minimize overhead for small accesses, but may adversely affect CPU utilization for large transfers.

File I/O is done directly to/from user-space buffers and memory mapped I/O may be performed with direct memory mappings that bypass the kernel page cache.

While the DAX property tends to result in data being transferred synchronously, it does not give the same guarantees as the O_SYNC flag (see open(2)), where data and the necessary metadata are transferred together.

A DAX file may support being mapped with the MAP_SYNC flag, which enables a program to use CPU cache flush instructions to persist CPU store operations without an ex? plicit fsync(2). See mmap(2) for more information.

STATX_ATTR_MOUNT_ROOT (since Linux 5.8)

The file is the root of a mount.

RETURN VALUE

On success, zero is returned. 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 path.

(See also path_resolution(7).)

EBADF path is relative but dirfd is neither AT_FDCWD nor a valid file descriptor.

EFAULT path or statxbuf points to a location outside the process's accessible address space

or is NULL (except since Linux 6.11 if AT_EMPTY_PATH is specified in flags, path is allowed to be NULL).

EINVAL Invalid flag specified in flags.

EINVAL Reserved flag specified in mask. (Currently, there is one such flag, designated by

the constant STATX__RESERVED, with the value 0x80000000U.)

ELOOP Too many symbolic links encountered while traversing the pathname.

ENAMETOOLONG

path is too long.

ENOENT A component of path does not exist, or path is an empty string and AT_EMPTY_PATH was not specified in flags.

ENOMEM Out of memory (i.e., kernel memory).

ENOTDIR

A component of the path prefix of path is not a directory or path is relative and dirfd is a file descriptor referring to a file other than a directory.

STANDARDS

Linux.

HISTORY

Linux 4.11, glibc 2.28.

SEE ALSO

ls(1), stat(1), access(2), chmod(2), chown(2), name_to_handle_at(2), readlink(2), stat(2), utime(2), proc(5), capabilities(7), inode(7), symlink(7)