stat, fstat, lstat - get file status

**Synopsis**

**#include <**[**sys/types.h**](https://linux.die.net/include/sys/types.h)**>  
#include <**[**sys/stat.h**](https://linux.die.net/include/sys/stat.h)**>  
#include <[unistd.h](https://linux.die.net/include/unistd.h)>**

**int stat(const char \****path***, struct stat \****buf***);  
int fstat(int** *fd***, struct stat \****buf***);  
int lstat(const char \****path***, struct stat \****buf***);**

**Description**

These functions return information about a file. No permissions are required on the file itself, but-in the case of **stat**() and **lstat**() - execute (search) permission is required on all of the directories in *path* that lead to the file.

**stat**() stats the file pointed to by *path* and fills in *buf*.

**lstat**() is identical to **stat**(), except that if *path* is a symbolic link, then the link itself is stat-ed, not the file that it refers to.

**fstat**() is identical to **stat**(), except that the file to be stat-ed is specified by the file descriptor *fd*.

All of these system calls return a *stat* structure, which contains the following fields:

struct stat {

dev\_t st\_dev; /\* ID of device containing file \*/

ino\_t st\_ino; /\* inode number \*/

mode\_t st\_mode; /\* protection \*/

nlink\_t st\_nlink; /\* number of hard links \*/

uid\_t st\_uid; /\* user ID of owner \*/

gid\_t st\_gid; /\* group ID of owner \*/

dev\_t st\_rdev; /\* device ID (if special file) \*/

off\_t st\_size; /\* total size, in bytes \*/

blksize\_t st\_blksize; /\* blocksize for file system I/O \*/

blkcnt\_t st\_blocks; /\* number of 512B blocks allocated \*/

time\_t st\_atime; /\* time of last access \*/

time\_t st\_mtime; /\* time of last modification \*/

time\_t st\_ctime; /\* time of last status change \*/

};

The *st\_dev* field describes the device on which this file resides. (The [**major**](https://linux.die.net/man/3/major)(3) and [**minor**](https://linux.die.net/man/3/minor)(3) macros may be useful to decompose the device ID in this field.)

The *st\_rdev* field describes the device that this file (inode) represents.

The *st\_size* field gives 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.

The *st\_blocks* field indicates the number of blocks allocated to the file, 512-byte units. (This may be smaller than *st\_size*/512 when the file has holes.)

The *st\_blksize* field gives the "preferred" blocksize for efficient file system I/O. (Writing to a file in smaller chunks may cause an inefficient read-modify-rewrite.)

Not all of the Linux file systems implement all of the time fields. Some file system types allow mounting in such a way that file and/or directory accesses do not cause an update of the *st\_atime* field. (See *noatime*, *nodiratime*, and *relatime* in [**mount**](https://linux.die.net/man/8/mount)(8), and related information in [**mount**](https://linux.die.net/man/2/mount)(2).) In addition, *st\_atime* is not updated if a file is opened with the **O\_NOATIME**; see [**open**](https://linux.die.net/man/2/open)(2).

The field *st\_atime* is changed by file accesses, for example, by **[execve](https://linux.die.net/man/2/execve)**(2), **[mknod](https://linux.die.net/man/2/mknod)**(2), [**pipe**](https://linux.die.net/man/2/pipe)(2), **[utime](https://linux.die.net/man/2/utime)**(2) and [**read**](https://linux.die.net/man/2/read)(2) (of more than zero bytes). Other routines, like **[mmap](https://linux.die.net/man/2/mmap)**(2), may or may not update *st\_atime*.

The field *st\_mtime* is changed by file modifications, for example, by **[mknod](https://linux.die.net/man/2/mknod)**(2), [**truncate**](https://linux.die.net/man/2/truncate)(2), **[utime](https://linux.die.net/man/2/utime)**(2) and [**write**](https://linux.die.net/man/2/write)(2) (of more than zero bytes). Moreover, *st\_mtime* of a directory is changed by the creation or deletion of files in that directory. The *st\_mtime* field is *not* changed for changes in owner, group, hard link count, or mode.

The field *st\_ctime* is changed by writing or by setting inode information (i.e., owner, group, link count, mode, etc.).

The following POSIX macros are defined to check the file type using the *st\_mode* field:

**S\_ISREG**(m)

is it a regular file?

**S\_ISDIR**(m)

directory?

**S\_ISCHR**(m)

character device?

**S\_ISBLK**(m)

block device?

**S\_ISFIFO**(m)

FIFO (named pipe)?

**S\_ISLNK**(m)

symbolic link? (Not in POSIX.1-1996.)

**S\_ISSOCK**(m)

socket? (Not in POSIX.1-1996.)

The following flags are defined for the *st\_mode* field:

The set-group-ID bit (**S\_ISGID**) has several special uses. For a directory it indicates that BSD semantics is to be used for that directory: files created there inherit their group ID from the directory, not from the effective group ID of the creating process, and directories created there will also get the **S\_ISGID** bit set. For a file that does not have the group execution bit (**S\_IXGRP**) set, the set-group-ID bit indicates mandatory file/record locking.

The sticky bit (**S\_ISVTX**) on a directory means that a file in that directory can be renamed or deleted only by the owner of the file, by the owner of the directory, and by a privileged process.

**Return Value**

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

**Errors**

**EACCES**

Search permission is denied for one of the directories in the path prefix of *path*. (See also **[path\_resolution](https://linux.die.net/man/7/path_resolution)**(7).)

**EBADF**

*fd* is bad.

**EFAULT**

Bad address.

**ELOOP**

Too many symbolic links encountered while traversing the path.

**ENAMETOOLONG**

*path* is too long.

**ENOENT**

A component of *path* does not exist, or *path* is an empty string.

**ENOMEM**

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

**ENOTDIR**

A component of the path prefix of *path* is not a directory.

**EOVERFLOW**

*path* or *fd* refers to a file whose size, inode number, or number of blocks cannot be represented in, respectively, the types *off\_t*, *ino\_t*, or *blkcnt\_t*. This error can occur when, for example, an application compiled on a 32-bit platform without *-D\_FILE\_OFFSET\_BITS=64* calls **stat**() on a file whose size exceeds *(1<<31)-1* bytes.

**Example**

The following program calls **stat**() and displays selected fields in the returned *stat* structure.

#include <[sys/types.h](https://linux.die.net/include/sys/types.h)>

#include <[sys/stat.h](https://linux.die.net/include/sys/stat.h)>

#include <[time.h](https://linux.die.net/include/time.h)>

#include <[stdio.h](https://linux.die.net/include/stdio.h)>

#include <[stdlib.h](https://linux.die.net/include/stdlib.h)>

int

main(int argc, char \*argv[])

{

struct stat sb;

if (argc != 2) {

fprintf(stderr, "Usage: %s <pathname>\n", argv[0]);

exit(EXIT\_FAILURE);

}

if (stat(argv[1], &sb) == -1) {

perror("stat");

exit(EXIT\_FAILURE);

}

printf("File type: ");

switch (sb.st\_mode & S\_IFMT) {

case S\_IFBLK: printf("block device\n"); break;

case S\_IFCHR: printf("character device\n"); break;

case S\_IFDIR: printf("directory\n"); break;

case S\_IFIFO: printf("FIFO/pipe\n"); break;

case S\_IFLNK: printf("symlink\n"); break;

case S\_IFREG: printf("regular file\n"); break;

case S\_IFSOCK: printf("socket\n"); break;

default: printf("unknown?\n"); break;

}

printf("I-node number: %ld\n", (long) sb.st\_ino);

printf("Mode: %lo (octal)\n",

(unsigned long) sb.st\_mode);

printf("Link count: %ld\n", (long) sb.st\_nlink);

printf("Ownership: UID=%ld GID=%ld\n",

(long) sb.st\_uid, (long) sb.st\_gid);

printf("Preferred I/O block size: %ld bytes\n",

(long) sb.st\_blksize);

printf("File size: %lld bytes\n",

(long long) sb.st\_size);

printf("Blocks allocated: %lld\n",

(long long) sb.st\_blocks);

printf("Last status change: %s", ctime(&sb.st\_ctime));

printf("Last file access: %s", ctime(&sb.st\_atime));

printf("Last file modification: %s", ctime(&sb.st\_mtime));

exit(EXIT\_SUCCESS);

}