

NAME

getcwd, getwd, get_current_dir_name – get current working directory

SYNOPSIS

```
#include <unistd.h>
```

```
char *getcwd(char *buf, size_t size);
```

```
char *getwd(char *buf);
```

```
char *get_current_dir_name(void);
```

Feature Test Macro Requirements for glibc (see **feature_test_macros(7)**):

```
get_current_dir_name():
    _GNU_SOURCE
```

```
getwd():
```

Since glibc 2.12:

```
(_XOPEN_SOURCE >= 500) && !(_POSIX_C_SOURCE >= 200809L)
```

```
    /* Glibc since 2.19: */ _DEFAULT_SOURCE
```

```
    /* Glibc versions <= 2.19: */ _BSD_SOURCE
```

Before glibc 2.12:

```
_BSD_SOURCE || _XOPEN_SOURCE >= 500
```

DESCRIPTION

These functions return a null-terminated string containing an absolute pathname that is the current working directory of the calling process. The pathname is returned as the function result and via the argument *buf*, if present.

The **getcwd()** function copies an absolute pathname of the current working directory to the array pointed to by *buf*, which is of length *size*.

If the length of the absolute pathname of the current working directory, including the terminating null byte, exceeds *size* bytes, NULL is returned, and *errno* is set to **ERANGE**; an application should check for this error, and allocate a larger buffer if necessary.

As an extension to the POSIX.1-2001 standard, glibc's **getcwd()** allocates the buffer dynamically using **malloc(3)** if *buf* is NULL. In this case, the allocated buffer has the length *size* unless *size* is zero, when *buf* is allocated as big as necessary. The caller should **free(3)** the returned buffer.

get_current_dir_name() will **malloc(3)** an array big enough to hold the absolute pathname of the current working directory. If the environment variable **PWD** is set, and its value is correct, then that value will be returned. The caller should **free(3)** the returned buffer.

getwd() does not **malloc(3)** any memory. The *buf* argument should be a pointer to an array at least **PATH_MAX** bytes long. If the length of the absolute pathname of the current working directory, including the terminating null byte, exceeds **PATH_MAX** bytes, NULL is returned, and *errno* is set to **ENAME_TOOLONG**. (Note that on some systems, **PATH_MAX** may not be a compile-time constant; furthermore, its value may depend on the filesystem, see **pathconf(3)**.) For portability and security reasons, use of **getwd()** is deprecated.

RETURN VALUE

On success, these functions return a pointer to a string containing the pathname of the current working directory. In the case **getcwd()** and **getwd()** this is the same value as *buf*.

On failure, these functions return NULL, and *errno* is set to indicate the error. The contents of the array pointed to by *buf* are undefined on error.

ERRORS**EACCES**

Permission to read or search a component of the filename was denied.

EFAULT

buf points to a bad address.

EINVAL

The *size* argument is zero and *buf* is not a null pointer.

EINVAL

getwd(): *buf* is NULL.

ENAMETOOLONG

getwd(): The size of the null-terminated absolute pathname string exceeds **PATH_MAX** bytes.

ENOENT

The current working directory has been unlinked.

ENOMEM

Out of memory.

ERANGE

The *size* argument is less than the length of the absolute pathname of the working directory, including the terminating null byte. You need to allocate a bigger array and try again.

ATTRIBUTES

For an explanation of the terms used in this section, see **attributes(7)**.

Interface	Attribute	Value
getcwd() , getwd()	Thread safety	MT-Safe
get_current_dir_name()	Thread safety	MT-Safe env

CONFORMING TO

getcwd() conforms to POSIX.1-2001. Note however that POSIX.1-2001 leaves the behavior of **getcwd()** unspecified if *buf* is NULL.

getwd() is present in POSIX.1-2001, but marked LEGACY. POSIX.1-2008 removes the specification of **getwd()**. Use **getcwd()** instead. POSIX.1-2001 does not define any errors for **getwd()**.

get_current_dir_name() is a GNU extension.

NOTES

Under Linux, these functions make use of the **getcwd()** system call (available since Linux 2.1.92). On older systems they would query */proc/self/cwd*. If both system call and proc filesystem are missing, a generic implementation is called. Only in that case can these calls fail under Linux with **EACCES**.

These functions are often used to save the location of the current working directory for the purpose of returning to it later. Opening the current directory (".") and calling **fchdir(2)** to return is usually a faster and more reliable alternative when sufficiently many file descriptors are available, especially on platforms other than Linux.

C library/kernel differences

On Linux, the kernel provides a **getcwd()** system call, which the functions described in this page will use if possible. The system call takes the same arguments as the library function of the same name, but is limited to returning at most **PATH_MAX** bytes. (Before Linux 3.12, the limit on the size of the returned pathname was the system page size. On many architectures, **PATH_MAX** and the system page size are both 4096 bytes, but a few architectures have a larger page size.) If the length of the pathname of the current working directory exceeds this limit, then the system call fails with the error **ENAMETOOLONG**. In this case, the library functions fall back to a (slower) alternative implementation that returns the full pathname.

Following a change in Linux 2.6.36, the pathname returned by the **getcwd()** system call will be prefixed with the string "(unreachable)" if the current directory is not below the root directory of the current process (e.g., because the process set a new filesystem root using **chroot(2)** without changing its current directory into the new root). Such behavior can also be caused by an unprivileged user by changing the current directory into another mount namespace. When dealing with pathname from untrusted sources, callers of the functions described in this page should consider checking whether the returned pathname starts with '/' or

'(' to avoid misinterpreting an unreachable path as a relative pathname.

BUGS

Since the Linux 2.6.36 change that added "(unreachable)" in the circumstances described above, the glibc implementation of **getcwd()** has failed to conform to POSIX and returned a relative pathname when the API contract requires an absolute pathname. With glibc 2.27 onwards this is corrected; calling **getcwd()** from such a pathname will now result in failure with **ENOENT**.

SEE ALSO

pwd(1), **chdir(2)**, **fchdir(2)**, **open(2)**, **unlink(2)**, **free(3)**, **malloc(3)**

COLOPHON

This page is part of release 5.02 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/>.