



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

\$ man unshare.2

unshare(2) System Calls Manual unshare(2)

NAME

unshare - disassociate parts of the process execution context

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

```
#define _GNU_SOURCE
```

```
#include <sched.h>
```

```
int unshare(int flags);
```

DESCRIPTION

unshare() allows a process (or thread) to disassociate parts of its execution context that are currently being shared with other processes (or threads). Part of the execution context, such as the mount namespace, is shared implicitly when a new process is created using fork(2) or vfork(2), while other parts, such as virtual memory, may be shared by explicit request when creating a process or thread using clone(2).

The main use of unshare() is to allow a process to control its shared execution context without creating a new process.

The flags argument is a bit mask that specifies which parts of the execution context should be unshared. This argument is specified by ORing together zero or more of the following constants:

CLONE_FILES

Reverse the effect of the clone(2) CLONE_FILES flag. Unshare the file descriptor table,

so that the calling process no longer shares its file descriptors with any other

process.

CLONE_FS

Reverse the effect of the clone(2) CLONE_FS flag. Unshare filesystem attributes, so that the calling process no longer shares its root directory (chroot(2)), current directory (chdir(2)), or umask (umask(2)) attributes with any other process.

CLONE_NEWCGROUP (since Linux 4.6)

This flag has the same effect as the clone(2) CLONE_NEWCGROUP flag. Unshare the cgroup namespace. Use of CLONE_NEWCGROUP requires the CAP_SYS_ADMIN capability.

CLONE_NEWIPC (since Linux 2.6.19)

This flag has the same effect as the clone(2) CLONE_NEWIPC flag. Unshare the IPC namespace, so that the calling process has a private copy of the IPC namespace which is not shared with any other process. Specifying this flag automatically implies CLONE_SYSVSEM as well. Use of CLONE_NEWIPC requires the CAP_SYS_ADMIN capability.

CLONE_NEWNET (since Linux 2.6.24)

This flag has the same effect as the clone(2) CLONE_NEWNET flag. Unshare the network namespace, so that the calling process is moved into a new network namespace which is not shared with any previously existing process. Use of CLONE_NEWNET requires the CAP_SYS_ADMIN capability.

CLONE_NEWNS

This flag has the same effect as the clone(2) CLONE_NEWNS flag. Unshare the mount namespace, so that the calling process has a private copy of its namespace which is not shared with any other process. Specifying this flag automatically implies CLONE_FS as well. Use of CLONE_NEWNS requires the CAP_SYS_ADMIN capability. For further information, see mount_namespaces(7).

CLONE_NEWPID (since Linux 3.8)

This flag has the same effect as the clone(2) CLONE_NEWPID flag. Unshare the PID namespace, so that the calling process has a new PID namespace for its children which is not shared with any previously existing process. The calling process is not moved into the new namespace. The first child created by the calling process will have the process ID 1 and will assume the role of init(1) in the new namespace. CLONE_NEWPID automatically implies CLONE_THREAD as well. Use of CLONE_NEWPID requires the CAP_SYS_ADMIN capability. For further information, see pid_namespaces(7).

CLONE_NEWTIME (since Linux 5.6)

Unshare the time namespace, so that the calling process has a new time namespace for its children which is not shared with any previously existing process. The calling process is not moved into the new namespace. Use of CLONE_NEWTIME requires the CAP_SYS_ADMIN capability. For further information, see time_namespaces(7).

CLONE_NEWUSER (since Linux 3.8)

This flag has the same effect as the clone(2) CLONE_NEWUSER flag. Unshare the user namespace, so that the calling process is moved into a new user namespace which is not shared with any previously existing process. As with the child process created by clone(2) with the CLONE_NEWUSER flag, the caller obtains a full set of capabilities in the new namespace.

CLONE_NEWUSER requires that the calling process is not threaded; specifying CLONE_NEWUSER automatically implies CLONE_THREAD. Since Linux 3.9, CLONE_NEWUSER also automatically implies CLONE_FS. CLONE_NEWUSER requires that the user ID and group ID of the calling process are mapped to user IDs and group IDs in the user namespace of the calling process at the time of the call.

For further information on user namespaces, see user_namespaces(7).

CLONE_NEWUTS (since Linux 2.6.19)

This flag has the same effect as the clone(2) CLONE_NEWUTS flag. Unshare the UTS IPC namespace, so that the calling process has a private copy of the UTS namespace which is not shared with any other process. Use of CLONE_NEWUTS requires the CAP_SYS_ADMIN capability.

CLONE_SYSVSEM (since Linux 2.6.26)

This flag reverses the effect of the clone(2) CLONE_SYSVSEM flag. Unshare System V semaphore adjustment (semadj) values, so that the calling process has a new empty semadj list that is not shared with any other process. If this is the last process that has a reference to the process's current semadj list, then the adjustments in that list are applied to the corresponding semaphores, as described in semop(2).

In addition, CLONE_THREAD, CLONE_SIGHAND, and CLONE_VM can be specified in flags if the caller is single threaded (i.e., it is not sharing its address space with another process or thread). In this case, these flags have no effect. (Note also that specifying CLONE_THREAD automatically implies CLONE_VM, and specifying CLONE_VM automatically implies CLONE_SIGHAND.) If the process is multithreaded, then the use of these flags results in an error.

If flags is specified as zero, then unshare() is a no-op; no changes are made to the calling

process's execution context.

RETURN VALUE

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

ERRORS

EINVAL An invalid bit was specified in flags.

EINVAL `CLONE_THREAD`, `CLONE_SIGHAND`, or `CLONE_VM` was specified in flags, and the caller is multithreaded.

EINVAL `CLONE_NEWIPC` was specified in flags, but the kernel was not configured with the `CONFIG_SYSVIPC` and `CONFIG_IPC_NS` options.

EINVAL `CLONE_NEWNET` was specified in flags, but the kernel was not configured with the `CONFIG_NET_NS` option.

EINVAL `CLONE_NEWPID` was specified in flags, but the kernel was not configured with the `CONFIG_PID_NS` option.

EINVAL `CLONE_NEWUSER` was specified in flags, but the kernel was not configured with the `CONFIG_USER_NS` option.

EINVAL `CLONE_NEWUTS` was specified in flags, but the kernel was not configured with the `CONFIG_UTS_NS` option.

EINVAL `CLONE_NEWPID` was specified in flags, but the process has previously called `unshare()` with the `CLONE_NEWPID` flag.

ENOMEM Cannot allocate sufficient memory to copy parts of caller's context that need to be unshared.

ENOSPC (since Linux 3.7)

`CLONE_NEWPID` was specified in flags, but the limit on the nesting depth of PID namespace would have been exceeded; see `pid_namespaces(7)`.

ENOSPC (since Linux 4.9; beforehand **EUSERS**)

`CLONE_NEWUSER` was specified in flags, and the call would cause the limit on the number of nested user namespaces to be exceeded. See `user_namespaces(7)`.

From Linux 3.11 to Linux 4.8, the error diagnosed in this case was **EUSERS**.

ENOSPC (since Linux 4.9)

One of the values in flags specified the creation of a new user namespace, but doing so would have caused the limit defined by the corresponding file in `/proc/sys/user` to be exceeded. For further details, see `namespaces(7)`.

EPERM The calling process did not have the required privileges for this operation.

EPERM CLONE_NEWUSER was specified in flags, but either the effective user ID or the effective group ID of the caller does not have a mapping in the parent namespace (see user_namespaces(7)).

EPERM (since Linux 3.9)

CLONE_NEWUSER was specified in flags and the caller is in a chroot environment (i.e., the caller's root directory does not match the root directory of the mount namespace in which it resides).

EUSERS (from Linux 3.11 to Linux 4.8)

CLONE_NEWUSER was specified in flags, and the limit on the number of nested user namespaces would be exceeded. See the discussion of the ENOSPC error above.

STANDARDS

Linux.

HISTORY

Linux 2.6.16.

NOTES

Not all of the process attributes that can be shared when a new process is created using clone(2) can be unshared using unshare(). In particular, as at kernel 3.8, unshare() does not implement flags that reverse the effects of CLONE_SIGHAND, CLONE_THREAD, or CLONE_VM. Such functionality may be added in the future, if required.

Creating all kinds of namespace, except user namespaces, requires the CAP_SYS_ADMIN capability. However, since creating a user namespace automatically confers a full set of capabilities, creating both a user namespace and any other type of namespace in the same unshare() call does not require the CAP_SYS_ADMIN capability in the original namespace.

EXAMPLES

The program below provides a simple implementation of the unshare(1) command, which unshares one or more namespaces and executes the command supplied in its command-line arguments. Here's an example of the use of this program, running a shell in a new mount namespace, and verifying that the original shell and the new shell are in separate mount namespaces:

```
$ readlink /proc/$$/ns/mnt;
mnt:[4026531840]
$ sudo ./unshare -m /bin/bash;
# readlink /proc/$$/ns/mnt;
```

mnt:[4026532325]

The differing output of the two readlink(1) commands shows that the two shells are in different mount namespaces.

Program source

```
/* unshare.c

A simple implementation of the unshare(1) command: unshare
namespaces and execute a command.

*/

#define _GNU_SOURCE

#include <err.h>
#include <sched.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

static void
usage(char *pname)
{
    fprintf(stderr, "Usage: %s [options] program [arg...]\n", pname);
    fprintf(stderr, "Options can be:\n");
    fprintf(stderr, "  -C unshare cgroup namespace\n");
    fprintf(stderr, "  -i unshare IPC namespace\n");
    fprintf(stderr, "  -m unshare mount namespace\n");
    fprintf(stderr, "  -n unshare network namespace\n");
    fprintf(stderr, "  -p unshare PID namespace\n");
    fprintf(stderr, "  -t unshare time namespace\n");
    fprintf(stderr, "  -u unshare UTS namespace\n");
    fprintf(stderr, "  -U unshare user namespace\n");
    exit(EXIT_FAILURE);
}

int
main(int argc, char *argv[])
{
    int flags, opt;
```

```

flags = 0;
while ((opt = getopt(argc, argv, "CimnpU")) != -1) {
    switch (opt) {
        case 'C': flags |= CLONE_NEWCGROUP;    break;
        case 'i': flags |= CLONE_NEWIPC;       break;
        case 'm': flags |= CLONE_NEWNS;        break;
        case 'n': flags |= CLONE_NEWNET;       break;
        case 'p': flags |= CLONE_NEWPID;       break;
        case 't': flags |= CLONE_NEWTIME;      break;
        case 'u': flags |= CLONE_NEWUTS;       break;
        case 'U': flags |= CLONE_NEWUSER;      break;
        default: usage(argv[0]);
    }
}
if (optind >= argc)
    usage(argv[0]);
if (unshare(flags) == -1)
    err(EXIT_FAILURE, "unshare");
execvp(argv[optind], &argv[optind]);
err(EXIT_FAILURE, "execvp");
}

```

SEE ALSO

unshare(1), clone(2), fork(2), kcmp(2), setns(2), vfork(2), namespaces(7)

Documentation/userspace-api/unshare.rst in the Linux kernel source tree