



Linux Ubuntu 26.04 Manual Pages on command 'unshare.1'

\$ man unshare.1

UNSHARE(1) User Commands UNSHARE(1)

NAME

unshare - run program in new namespaces

SYNOPSIS

unshare [options] [program [arguments]]

DESCRIPTION

The unshare command creates new namespaces (as specified by the command-line options described below) and then executes the specified program. If program is not given, then "\${SHELL}" is run (default: /bin/sh).

By default, a new namespace persists only as long as it has member processes. A new namespace can be made persistent even when it has no member processes by bind mounting /proc/pid/ns/type files to a filesystem path. A namespace that has been made persistent in this way can subsequently be entered with nsenter(1) even after the program terminates (except PID namespaces where a permanently running init process is required). Once a persistent namespace is no longer needed, it can be unpersisted by using umount(8) to remove the bind mount. See the EXAMPLES section for more details.

unshare since util-linux version 2.36 uses /proc/[pid]/ns/pid_for_children and /proc/[pid]/ns/time_for_children files for persistent PID and TIME namespaces. This change requires Linux kernel 4.17 or newer.

The following types of namespaces can be created with unshare:

mount namespace

Mounting and unmounting filesystems will not affect the rest of the system, except for filesystems which are explicitly marked as shared (with mount --make-shared; see

/proc/self/mountinfo or findmnt -o+PROPAGATION for the shared flags). For further details, see mount_namespaces(7).

unshare since util-linux version 2.27 automatically sets propagation to private in a new mount namespace to make sure that the new namespace is really unshared. It's possible to disable this feature with option --propagation unchanged. Note that private is the kernel default.

UTS namespace

Setting hostname or domainname will not affect the rest of the system. For further details, see uts_namespaces(7).

IPC namespace

The process will have an independent namespace for POSIX message queues as well as System V message queues, semaphore sets and shared memory segments. For further details, see ipc_namespaces(7).

network namespace

The process will have independent IPv4 and IPv6 stacks, IP routing tables, firewall rules, the /proc/net and /sys/class/net directory trees, sockets, etc. For further details, see network_namespaces(7).

PID namespace

Children will have a distinct set of PID-to-process mappings from their parent. For further details, see pid_namespaces(7).

cgroup namespace

The process will have a virtualized view of /proc/self/cgroup, and new cgroup mounts will be rooted at the namespace cgroup root. For further details, see cgroup_namespaces(7).

user namespace

The process will have a distinct set of UIDs, GIDs and capabilities. For further details, see user_namespaces(7).

time namespace

The process can have a distinct view of CLOCK_MONOTONIC and/or CLOCK_BOOTTIME which can be changed using /proc/self/timens_offsets. For further details, see time_namespaces(7).

OPTIONS

-i, --ipc[=file]

Create a new IPC namespace. If file is specified, then the namespace is made persistent

by creating a bind mount at file.

-m, --mount[=file]

Create a new mount namespace. If file is specified, then the namespace is made persistent by creating a bind mount at file. Note that file must be located on a mount whose propagation type is not shared (or an error results). Use the command `findmnt -o+PROPAGATION` when not sure about the current setting. See also the examples below.

-n, --net[=file]

Create a new network namespace. If file is specified, then the namespace is made persistent by creating a bind mount at file.

-p, --pid[=file]

Create a new PID namespace. If file is specified, then the namespace is made persistent by creating a bind mount at file. (Creation of a persistent PID namespace will fail if the `--fork` option is not also specified.)

See also the `--fork` and `--mount-proc` options.

-u, --uts[=file]

Create a new UTS namespace. If file is specified, then the namespace is made persistent by creating a bind mount at file.

-U, --user[=file]

Create a new user namespace. If file is specified, then the namespace is made persistent by creating a bind mount at file.

-C, --cgroup[=file]

Create a new cgroup namespace. If file is specified, then the namespace is made persistent by creating a bind mount at file.

-T, --time[=file]

Create a new time namespace. If file is specified, then the namespace is made persistent by creating a bind mount at file. The `--monotonic` and `--boottime` options can be used to specify the corresponding offset in the time namespace.

-f, --fork

Fork the specified program as a child process of `unshare` rather than running it directly. This is useful when creating a new PID namespace. Note that when `unshare` is waiting for the child process, then it ignores `SIGINT` and `SIGTERM` and does not forward any signals to the child. It is necessary to send signals to the child process.

--keep-caps

When the `--user` option is given, ensure that capabilities granted in the user namespace are preserved in the child process.

`--kill-child[=signame]`

When `unshare` terminates, have `signame` be sent to the forked child process. Combined with `--pid` this allows for an easy and reliable killing of the entire process tree below `unshare`. If not given, `signame` defaults to `SIGKILL`. This option implies `--fork`.

`--mount-proc[=mountpoint]`

Just before running the program, mount the `proc` filesystem at `mountpoint` (default is `/proc`). This is useful when creating a new PID namespace. It also implies creating a new mount namespace since the `/proc` mount would otherwise mess up existing programs on the system. The new `proc` filesystem is explicitly mounted as private (with `MS_PRIVATE|MS_REC`).

`--mount-binfmt[=mountpoint]`

Just before running the program, mount the `binfmt_misc` filesystem at `mountpoint` (default is `/proc/sys/fs/binfmt_misc`). It also implies creating a new mount namespace since the `binfmt_misc` mount would otherwise mess up existing programs on the system. The new `binfmt_misc` filesystem is explicitly mounted as private (with `MS_PRIVATE|MS_REC`).

`--map-user uid|name`

Run the program only after the current effective user ID has been mapped to `uid`. If this option is specified multiple times, the last occurrence takes precedence. This option implies `--user`.

`--map-users inneruid:outeruid:count|auto|subids|all`

Run the program only after the block of user IDs of size `count` beginning at `outeruid` has been mapped to the block of user IDs beginning at `inneruid`. This mapping is created with `newuidmap(1)` if `unshare` was run unprivileged. If the range of user IDs overlaps with the mapping specified by `--map-user`, then a "hole" will be removed from the mapping. This may result in the highest user ID of the mapping not being mapped. Use `--map-users` multiple times to map more than one block of user IDs. The special value `auto` will map the first block of user IDs owned by the effective user from `/etc/subuid` to a block starting at user ID 0. The special value `subids` will identity map the same block. The special value `all` will create a pass-through map for every user ID available in the parent namespace. This option implies `--user`.

Before `util-linux` version 2.39, this option expected a comma-separated argument of the

form outeruid,inneruid,count but that format is now deprecated for consistency with the ordering used in /proc/[pid]/uid_map and the X-mount.idmap mount option.

`--map-group gid|name`

Run the program only after the current effective group ID has been mapped to gid. If this option is specified multiple times, the last occurrence takes precedence. This option implies `--setgroups=deny` and `--user`.

`--map-groups innergid:outergid:count[auto|subids|all]`

Run the program only after the block of group IDs of size count beginning at outergid has been mapped to the block of group IDs beginning at innergid. This mapping is created with `newgidmap(1)` if `unshare` was run unprivileged. If the range of group IDs overlaps with the mapping specified by `--map-group`, then a "hole" will be removed from the mapping. This may result in the highest group ID of the mapping not being mapped. Use `--map-groups` multiple times to map more than one block of group IDs. The special value `auto` will map the first block of user IDs owned by the effective user from `/etc/subgid` to a block starting at group ID 0. The special value `subids` will identity map the same block. The special value `all` will create a pass-through map for every group ID available in the parent namespace. This option implies `--user`.

Before `util-linux` version 2.39, this option expected a comma-separated argument of the form `outergid,innergid,count` but that format is now deprecated for consistency with the ordering used in `/proc/[pid]/gid_map` and the `X-mount.idmap` mount option.

`--map-auto`

Map the first block of user IDs owned by the effective user from `/etc/subuid` to a block starting at user ID 0. In the same manner, also map the first block of group IDs owned by the effective group from `/etc/subgid` to a block starting at group ID 0. This option is intended to handle the common case where the first block of subordinate user and group IDs can map the whole user and group ID space. This option is equivalent to specifying `--map-users=auto` and `--map-groups=auto`.

`--map-subids`

Identity map the first block of user IDs owned by the effective user from `/etc/subuid`. In the same manner, also identity map the first block of group IDs owned by the effective group from `/etc/subgid`. This option is equivalent to specifying `--map-users=subids` and `--map-groups=subids`.

`-r, --map-root-user`

Run the program only after the current effective user and group IDs have been mapped to the superuser UID and GID in the newly created user namespace. This makes it possible to conveniently gain capabilities needed to manage various aspects of the newly created namespaces (such as configuring interfaces in the network namespace or mounting filesystems in the mount namespace) even when run unprivileged. As a mere convenience feature, it does not support more sophisticated use cases, such as mapping multiple ranges of UIDs and GIDs. This option implies `--setgroups=deny` and `--user`. This option is equivalent to `--map-user=0 --map-group=0`.

`-c, --map-current-user`

Run the program only after the current effective user and group IDs have been mapped to the same UID and GID in the newly created user namespace. This option implies `--setgroups=deny` and `--user`. This option is equivalent to `--map-user=$(id -ru) --map-group=$(id -rg)`.

`--owner uid:gid`

Set the owner user and group when creating a user namespace. These determine which user in the parent namespace has `CAP_SYS_ADMIN` in the new child namespace and can `setns(2)` into it. This option allows a privileged user to create a namespace on behalf of an unprivileged one, using its privileges to map ids and/or bind mount the namespace into the filesystem. It implies `--user`.

`--propagation private|shared|slave|unchanged`

Recursively set the mount propagation flag in the new mount namespace. The default is to set the propagation to private. It is possible to disable this feature with the argument `unchanged`. The option is silently ignored when the mount namespace (`--mount`) is not requested.

`--setgroups allow|deny`

Allow or deny the `setgroups(2)` system call in a user namespace.

To be able to call `setgroups(2)`, the calling process must at least have `CAP_SETGID`. But since Linux 3.19 a further restriction applies: the kernel gives permission to call `setgroups(2)` only after the GID map (`/proc/pid*/gid_map*`) has been set. The GID map is writable by root when `setgroups(2)` is enabled (i.e., `allow`, the default), and the GID map becomes writable by unprivileged processes when `setgroups(2)` is permanently disabled (with `deny`).

`-R, --root dir`

run the command with root directory set to dir.

-w, --wd dir

change working directory to dir.

-S, --setuid uid

Set the user ID which will be used in the entered namespace.

-G, --setgid gid

Set the group ID which will be used in the entered namespace and drop supplementary groups.

-l, --load-interp string

Load binfmt_misc definition in the namespace (implies --mount-binfmt). The string argument is :name:type:offset:magic:mask:interpreter:flags. For more details about new binary type registration see

<https://www.kernel.org/doc/Documentation/admin-guide/binfmt-misc.rst>. To manage the F flag in flags with --root parameter, binfmt_misc is mounted twice, once before the chroot to load the interpreter from the caller filesystem and once after to make it available from the chroot userspace.

--monotonic offset

Set the offset of CLOCK_MONOTONIC which will be used in the entered time namespace. This option requires unsharing a time namespace with --time.

--boottime offset

Set the offset of CLOCK_BOOTTIME which will be used in the entered time namespace. This option requires unsharing a time namespace with --time.

-h, --help

Display help text and exit.

-V, --version

Display version and exit.

NOTES

The proc and sysfs filesystems mounting as root in a user namespace have to be restricted so that a less privileged user cannot get more access to sensitive files that a more privileged user made unavailable. In short the rule for proc and sysfs is as close to a bind mount as possible.

EXAMPLES

The following command creates a PID namespace, using --fork to ensure that the executed

command is performed in a child process that (being the first process in the namespace) has PID 1. The --mount-proc option ensures that a new mount namespace is also simultaneously created and that a new proc(5) filesystem is mounted that contains information corresponding to the new PID namespace. When the readlink(1) command terminates, the new namespaces are automatically torn down.

```
# unshare --fork --pid --mount-proc readlink /proc/self
1
```

As an unprivileged user, create a new user namespace where the user's credentials are mapped to the root IDs inside the namespace:

```
$ id -u; id -g
1000
1000
$ unshare --user --map-root-user \
    sh -c 'whoami; cat /proc/self/uid_map /proc/self/gid_map'
root
0    1000    1
0    1000    1
```

As an unprivileged user, create a user namespace where the first 65536 IDs are all mapped, and the user's credentials are mapped to the root IDs inside the namespace. The map is determined by the subordinate IDs assigned in subuid(5) and subgid(5). Demonstrate this mapping by creating a file with user ID 1 and group ID 1. For brevity, only the user ID mappings are shown:

```
$ id -u
1000
$ cat /etc/subuid
1000:100000:65536
$ unshare --user --map-auto --map-root-user
# id -u
0
# cat /proc/self/uid_map
0    1000    1
1    100000  65535
# touch file; chown 1:1 file
```

```
# ls -ln --time-style=+ file
-rw-r--r-- 1 1 1 0 file

# exit

$ ls -ln --time-style=+ file
-rw-r--r-- 1 100000 100000 0 file
```

The first of the following commands creates a new persistent UTS namespace and modifies the hostname as seen in that namespace. The namespace is then entered with nsenter(1) in order to display the modified hostname; this step demonstrates that the UTS namespace continues to exist even though the namespace had no member processes after the unshare command terminated. The namespace is then destroyed by removing the bind mount.

```
# touch /root/uts-ns

# unshare --uts=/root/uts-ns hostname FOO

# nsenter --uts=/root/uts-ns hostname
FOO

# umount /root/uts-ns
```

The following commands establish a persistent mount namespace referenced by the bind mount /root/namespaces/mnt. In order to ensure that the creation of that bind mount succeeds, the parent directory (/root/namespaces) is made a bind mount whose propagation type is not shared.

```
# mount --bind /root/namespaces /root/namespaces

# mount --make-private /root/namespaces

# touch /root/namespaces/mnt

# unshare --mount=/root/namespaces/mnt
```

The following commands demonstrate the use of the --kill-child option when creating a PID namespace, in order to ensure that when unshare is killed, all of the processes within the PID namespace are killed.

```
# set +m          # Don't print job status messages

# unshare --pid --fork --mount-proc --kill-child -- \
    bash --norc -c '(sleep 555 &) && (ps a &) && sleep 999' &

[1] 53456

#  PID TTY    STAT  TIME COMMAND
    1 pts/3  S+    0:00 sleep 999
    3 pts/3  S+    0:00 sleep 555
```

```
# ps h -o 'comm' $!    # Show that background job is unshare(1)
unshare
# kill $!              # Kill unshare(1)
# pidof sleep
```

```
# unshare --pid --fork --mount-proc -- \
    bash --norc -c '(sleep 555 &) && (ps a &) && sleep 999' &
```

```
# PID TTY STAT TIME COMMAND
1 pts/3 S+ 0:00 sleep 999
3 pts/3 S+ 0:00 sleep 555
5 pts/3 R+ 0:00 ps a
```

53482 53480

```
# uptime -p          # Show uptime in initial time namespace
up 21 hours, 30 minutes

# unshare --time --fork --boottime 300000000 uptime -p
up 9 years, 28 weeks, 1 day, 2 hours, 50 minutes
```

```
$ unshare --map-root-user --fork --pid
```

The load-interp parameter can be read as following

qemu-ppc

is the name of the new file created below `/proc/sys/fs/binfmt_misc` to register the interpreter

M

defines the interpreter for a given type of magic number

`\x7fELF\x01\x02\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x02\x00\x1`

is the magic number to recognize the file to interpret (in this case, the ELF header for PPC32)

`\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff`

the mask to apply to the magic number

`/bin/qemu-ppc-static`

the interpreter to use with the file

OCF

the file is open by the kernel with credential and security tokens of the file itself and loaded as soon as we register it.

AUTHORS

Mikhail Gusarov <dottedmag@dottedmag.net>, Karel Zak <kzak@redhat.com>

SEE ALSO

`newuidmap(1)`, `newgidmap(1)`, `clone(2)`, `unshare(2)`, `namespaces(7)`, `mount(8)`

REPORTING BUGS

For bug reports, use the issue tracker <<https://github.com/util-linux/util-linux/issues>>.

AVAILABILITY

The `unshare` command is part of the `util-linux` package which can be downloaded from Linux Kernel Archive <<https://www.kernel.org/pub/linux/utils/util-linux/>>.

util-linux 2.41.3

2026-03-13

UNSHARE(1)