Rocky Enterprise Linux 9.2 Manual Pages on command 'unshare.1'

C:\>man unshare.1

UNSHARE(1) User Commands UNSHARE(1)

NAME

unshare - run program with some namespaces unshared from parent

SYNOPSIS

unshare [options] [program [arguments]]

DESCRIPTION

Unshares the indicated namespaces from the parent process and then executes the specified program. If program is not given, then ``${SHELL}'' is run (default: /bin/sh).

The namespaces can optionally be made persistent by bind mounting /proc/pid/ns/type files to a filesystem path and entered with nsenter(1) even after the program ter? minates (except PID namespaces where permanently running init process is required). Once a persistent namespace is no longer needed, it can be unpersisted with umount(8). See the EXAMPLES section for more details.

The namespaces to be unshared are indicated via options. Unshareable namespaces are:

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) and the discus? sion of the CLONE_NEWNS flag in clone(2).

unshare since util-linux version 2.27 automatically sets propagation to pri?
vate 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 namespaces(7) and the discussion of the CLONE_NEWUTS flag in clone(2).

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 namespaces(7) and the discussion of the CLONE_NEWIPC flag in clone(2).

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 namespaces(7) and the discussion of the CLONE_NEWNET flag in clone(2).

PID namespace

Children will have a distinct set of PID-to-process mappings from their par?
ent. For further details, see pid_namespaces(7) and the discussion of the CLONE_NEWPID flag in clone(2).

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 de?
tails, see cgroup_namespaces(7) and the discussion of the CLONE_NEWCGROUP flag in clone(2).

user namespace

The process will have a distinct set of UIDs, GIDs and capabilities. For further details, see user_namespaces(7) and the discussion of the CLONE_NEWUSER flag in clone(2).

OPTIONS

-i, --ipc[=file]

Unshare the IPC namespace. If file is specified, then a persistent name? space is created by a bind mount.

-m, --mount[=file]

Unshare the mount namespace. If file is specified, then a persistent name? space is created by a bind mount. Note that file has to be located on a filesystem with the propagation flag set to private. Use the command find? mnt -o+PROPAGATION when not sure about the current setting. See also the examples below.

-n, --net[=file]

Unshare the network namespace. If file is specified, then a persistent namespace is created by a bind mount.

-p, --pid[=file]

Unshare the PID namespace. If file is specified then persistent namespace is created by a bind mount. See also the --fork and --mount-proc options.

-u, --uts[=file]

Unshare the UTS namespace. If file is specified, then a persistent name? space is created by a bind mount.

-U, --user[=file]

Unshare the user namespace. If file is specified, then a persistent name? space is created by a bind mount.

-C, --cgroup[=file]

Unshare the cgroup namespace. If file is specified then persistent namespace is created by bind mount.

-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.

--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 en? tire 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 oth? erwise mess up existing programs on the system. The new proc filesystem is explicitly mounted as private (with MS_PRIVATE|MS_REC).

-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 name? space. 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.

--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 sup? plementary groups.

-V, --version

Display version information and exit.

-h, --help

Display help text and exit.

NOTES

The proc and sysfs filesystems mounting as root in a user namespace have to be re? stricted so that a less privileged user can not 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

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

Establish a PID namespace, ensure we're PID 1 in it against a newly mounted procfs instance.

```
$ unshare --map-root-user --user sh -c whoami
root
```

Establish a user namespace as an unprivileged user with a root user within it.

```
# touch /root/uts-ns
# unshare --uts=/root/uts-ns hostname FOO
# nsenter --uts=/root/uts-ns hostname
FOO
# umount /root/uts-ns
```

Establish a persistent UTS namespace, and modify the hostname. The name? space is then entered with nsenter. The namespace is destroyed by unmount? ing the bind reference.

```
# mount --bind /root/namespaces /root/namespaces
# mount --make-private /root/namespaces
# touch /root/namespaces/mnt
# unshare --mount=/root/namespaces/mnt
```

Establish a persistent mount namespace referenced by the bind mount /root/namespaces/mnt. This example shows a portable solution, because it makes sure that the bind mount is created on a shared filesystem.

```
# unshare -pf --kill-child -- bash -c (sleep 999 &) && sleep 1000 &
# pid=$!
# kill $pid
```

Reliable killing of subprocesses of the program. When unshare gets killed, everything below it gets killed as well. Without it, the children of pro? gram would have orphaned and been re-parented to PID 1.

SEE ALSO

clone(2), unshare(2), namespaces(7), mount(8)

AUTHORS

Mikhail Gusarov ?dottedmag@dottedmag.net?

Karel Zak ?kzak@redhat.com?

AVAILABILITY

The unshare command is part of the util-linux package and is available from https://www.kernel.org/pub/linux/utils/util-linux/.