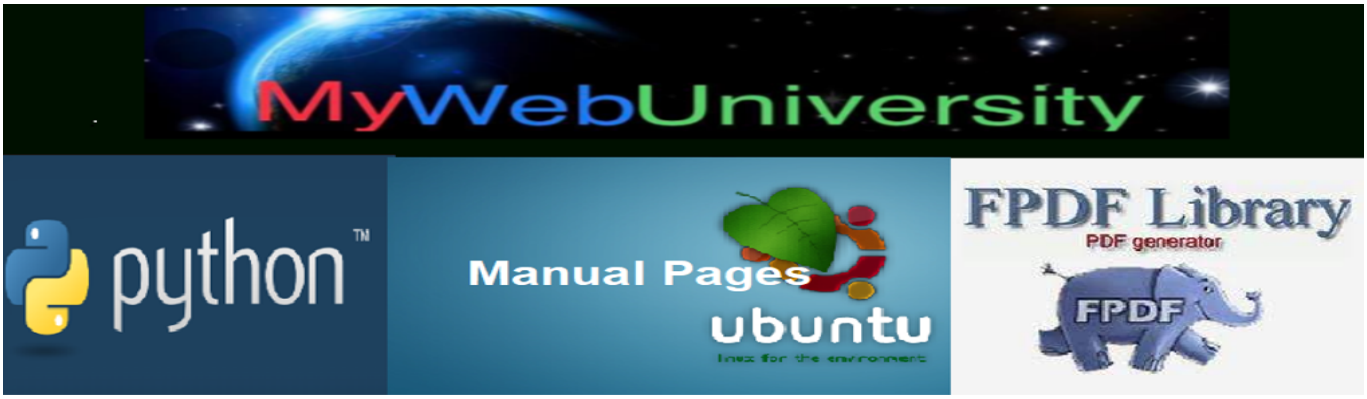

Rocky Enterprise Linux 9.2 Manual Pages on command 'unshare.1'

$ 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]

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

-m, --mount[=file]

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

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 a persistent namespace is created by a bind mount. (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]

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

-U, --user[=file]

Unshare the user namespace. If file is specified, then a persistent namespace 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.

-T, --time[=file]

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

--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-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.

-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).

--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.

--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.

-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 restricted 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

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 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
```

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
     5 pts/3    R+     0:00 ps a
# ps h -o 'comm' $! # Show that background job is unshare(1)
unshare
# kill $! # Kill unshare(1)
# pidof sleep
```

The pidof(1) command prints no output, because the sleep processes have been killed. More precisely, when the sleep process that has PID 1 in the namespace (i.e., the namespace?s init process) was killed, this caused all other processes in the namespace to be killed.
By contrast, a similar series of commands where the --kill-child option is not used shows that when unshare terminates, the processes in the PID namespace are not killed:

```
# unshare --pid --fork --mount-proc -- \
      bash --norc -c "(sleep 555 &) && (ps a &) && sleep 999" &
[1] 53479
```

```
# 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
# kill $!
# pidof sleep
53482 53480
```

The following example demonstrates the creation of a time namespace where the boottime clock is set to a point several years in the past:

```
# 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
```

AUTHORS

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

SEE ALSO

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

REPORTING BUGS

For bug reports, use the issue tracker at https://github.com/karelzak/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/>.