NAME

tmpfiles.d − Configuration for creation, deletion and cleaning of volatile and temporary files

SYNOPSIS

/etc/tmpfiles.d/*.conf
/run/tmpfiles.d/*.conf
/usr/lib/tmpfiles.d/*.conf

~/.config/user−tmpfiles.d/*.conf
$XDG_RUNTIME_DIR/user−tmpfiles.d/*.conf
~/.local/share/user−tmpfiles.d/*.conf
...
/usr/share/user−tmpfiles.d/*.conf

DESCRIPTION

tmpfiles.d configuration files provide a generic mechanism to define the *creation* of regular files, directories, pipes, and device nodes, adjustments to their *access mode, ownership, attributes, quota assignments, and contents*, and finally their time−based *removal*. It is mostly commonly used for volatile and temporary files and directories (such as those located under /run, /tmp, /var/tmp, the API file systems such as /sys or /proc, as well as some other directories below /var).

systemd−tmpfiles uses this configuration to create volatile files and directories during boot and to do periodic cleanup afterwards. See **systemd-tmpfiles**(5) for the description of systemd−tmpfiles−setup.service, systemd−tmpfiles−cleanup.service, and associated units.

System daemons frequently require private runtime directories below /run to store communication sockets and similar. For these, is is better to use *RuntimeDirectory*= in their unit files (see **systemd.exec**(5) for details), if the flexibility provided by tmpfiles.d is not required. The advantages are that the configuration required by the unit is centralized in one place, and that the lifetime of the directory is tied to the lifetime of the service itself. Similarly, *StateDirectory*=, *CacheDirectory*=, *LogsDirectory*=, and *ConfigurationDirectory*= should be used to create directories under /var/lib/, /var/cache/, /var/log/, and /etc/. tmpfiles.d should be used for files whose lifetime is independent of any service or requires more complicated configuration.

CONFIGURATION DIRECTORIES AND PRECEDENCE

Each configuration file shall be named in the style of *package*.conf or *package−part*.conf. The second variant should be used when it is desirable to make it easy to override just this part of configuration.

Files in /etc/tmpfiles.d override files with the same name in /usr/lib/tmpfiles.d and /run/tmpfiles.d. Files in /run/tmpfiles.d override files with the same name in /usr/lib/tmpfiles.d. Packages should install their configuration files in /usr/lib/tmpfiles.d. Files in /etc/tmpfiles.d are reserved for the local administrator, who may use this logic to override the configuration files installed by vendor packages. All configuration files are sorted by their filename in lexicographic order, regardless of which of the directories they reside in. If multiple files specify the same path, the entry in the file with the lexicographically earliest name will be applied. All other conflicting entries will be logged as errors. When two lines are prefix path and suffix path of each other, then the prefix line is always created first, the suffix later (and if removal applies to the line, the order is reversed: the suffix is removed first, the prefix later). Lines that take globs are applied after those accepting no globs. If multiple operations shall be applied on the same file (such as ACL, xattr, file attribute adjustments), these are always done in the same fixed order. Except for those cases, the files/directories are processed in the order they are listed.

If the administrator wants to disable a configuration file supplied by the vendor, the recommended way is to place a symlink to /dev/null in /etc/tmpfiles.d/ bearing the same filename.

CONFIGURATION FILE FORMAT

The configuration format is one line per path containing type, path, mode, ownership, age, and argument fields:

```
#Type Path       Mode User Group Age Argument
d    /run/user   0755 root root  10d -
L    /tmp/foobar -    -    -     -   /dev/null
```

Fields may be enclosed within quotes and contain C−style escapes.

Type

The type consists of a single letter and optionally an exclamation mark and/or minus sign.

The following line types are understood:

f

Create a file if it does not exist yet. If the argument parameter is given and the file did not exist yet, it will be written to the file. Does not follow symlinks.

F

Create or truncate a file. If the argument parameter is given, it will be written to the file. Does not follow symlinks.

w

Write the argument parameter to a file, if the file exists. Lines of this type accept shell−style globs in place of normal path names. The argument parameter will be written without a trailing newline. C−style backslash escapes are interpreted. Follows symlinks.

d

Create a directory. The mode and ownership will be adjusted if specified. Contents of this directory are subject to time based cleanup if the age argument is specified.

D

Similar to *d*, but in addition the contents of the directory will be removed when **−−remove** is used.

e

Adjust the mode and ownership of existing directories and remove their contents based on age. Lines of this type accept shell−style globs in place of normal path names. Contents of the directories are subject to time based cleanup if the age argument is specified. If the age argument is "0", contents will be unconditionally deleted every time **systemd−tmpfiles −−clean** is run.

For this entry to be useful, at least one of the mode, user, group, or age arguments must be specified, since otherwise this entry has no effect. As an exception, an entry with no effect may be useful when combined with *!*, see the examples.

v

Create a subvolume if the path does not exist yet, the file system supports subvolumes (btrfs), and the system itself is installed into a subvolume (specifically: the root directory **/** is itself a subvolume). Otherwise, create a normal directory, in the same way as *d*.

A subvolume created with this line type is not assigned to any higher−level quota group. For that, use *q* or *Q*, which allow creating simple quota group hierarchies, see below.

q

Create a subvolume or directory the same as *v*, but assign the subvolume to the same higher−level quota groups as the parent. This ensures that higher−level limits and accounting applied to the parent subvolume also include the specified subvolume. On non−btrfs file systems, this line type is identical to *d*.

If the subvolume already exists, no change to the quota hierarchy is made, regardless of whether the subvolume is already attached to a quota group or not. Also see *Q* below. See **btrfs-qgroup**(8) for details about the btrfs quota group concept.

Q

Create the subvolume or directory the same as *v*, but assign the new subvolume to a new leaf quota group. Instead of copying the higher−level quota group assignments from the parent as is done with *q*,

the lowest quota group of the parent subvolume is determined that is not the leaf quota group. Then, an "intermediary" quota group is inserted that is one level below this level, and shares the same ID part as the specified subvolume. If no higher−level quota group exists for the parent subvolume, a new quota group at level 255 sharing the same ID as the specified subvolume is inserted instead. This new intermediary quota group is then assigned to the parent subvolume's higher−level quota groups, and the specified subvolume's leaf quota group is assigned to it.

Effectively, this has a similar effect as *q*, however introduces a new higher−level quota group for the specified subvolume that may be used to enforce limits and accounting to the specified subvolume and children subvolume created within it. Thus, by creating subvolumes only via *q* and *Q*, a concept of "subtree quotas" is implemented. Each subvolume for which *Q* is set will get a "subtree" quota group created, and all child subvolumes created within it will be assigned to it. Each subvolume for which *q* is set will not get such a "subtree" quota group, but it is ensured that they are added to the same "subtree" quota group as their immediate parents.

It is recommended to use *Q* for subvolumes that typically contain further subvolumes, and where it is desirable to have accounting and quota limits on all child subvolumes together. Examples for *Q* are typically /home or /var/lib/machines. In contrast, *q* should be used for subvolumes that either usually do not include further subvolumes or where no accounting and quota limits are needed that apply to all child subvolumes together. Examples for *q* are typically /var or /var/tmp.

As with *q*, *Q* has no effect on the quota group hierarchy if the subvolume already exists, regardless of whether the subvolume already belong to a quota group or not.

p, *p+*
Create a named pipe (FIFO) if it does not exist yet. If suffixed with + and a file already exists where the pipe is to be created, it will be removed and be replaced by the pipe.

L, *L+*
Create a symlink if it does not exist yet. If suffixed with + and a file or directory already exists where the symlink is to be created, it will be removed and be replaced by the symlink. If the argument is omitted, symlinks to files with the same name residing in the directory /usr/share/factory/ are created. Note that permissions and ownership on symlinks are ignored.

c, *c+*
Create a character device node if it does not exist yet. If suffixed with + and a file already exists where the device node is to be created, it will be removed and be replaced by the device node. It is recommended to suffix this entry with an exclamation mark to only create static device nodes at boot, as udev will not manage static device nodes that are created at runtime.

b, *b+*
Create a block device node if it does not exist yet. If suffixed with + and a file already exists where the device node is to be created, it will be removed and be replaced by the device node. It is recommended to suffix this entry with an exclamation mark to only create static device nodes at boot, as udev will not manage static device nodes that are created at runtime.

C
Recursively copy a file or directory, if the destination files or directories do not exist yet or the destination directory is empty. Note that this command will not descend into subdirectories if the destination directory already exists and is not empty. Instead, the entire copy operation is skipped. If the argument is omitted, files from the source directory /usr/share/factory/ with the same name are copied. Does not follow symlinks.

x
Ignore a path during cleaning. Use this type to exclude paths from clean−up as controlled with the Age parameter. Note that lines of this type do not influence the effect of *r* or *R* lines. Lines of this type accept shell−style globs in place of normal path names.

X

Ignore a path during cleaning. Use this type to exclude paths from clean−up as controlled with the Age parameter. Unlike *x*, this parameter will not exclude the content if path is a directory, but only directory itself. Note that lines of this type do not influence the effect of *r* or *R* lines. Lines of this type accept shell−style globs in place of normal path names.

r

Remove a file or directory if it exists. This may not be used to remove non−empty directories, use *R* for that. Lines of this type accept shell−style globs in place of normal path names. Does not follow symlinks.

R

Recursively remove a path and all its subdirectories (if it is a directory). Lines of this type accept shell−style globs in place of normal path names. Does not follow symlinks.

z

Adjust the access mode, user and group ownership, and restore the SELinux security context of a file or directory, if it exists. Lines of this type accept shell−style globs in place of normal path names. Does not follow symlinks.

Z

Recursively set the access mode, user and group ownership, and restore the SELinux security context of a file or directory if it exists, as well as of its subdirectories and the files contained therein (if applicable). Lines of this type accept shell−style globs in place of normal path names. Does not follow symlinks.

t

Set extended attributes. Lines of this type accept shell−style globs in place of normal path names. This can be useful for setting SMACK labels. Does not follow symlinks.

T

Recursively set extended attributes. Lines of this type accept shell−style globs in place of normal path names. This can be useful for setting SMACK labels. Does not follow symlinks.

h

Set file/directory attributes. Lines of this type accept shell−style globs in place of normal path names.

The format of the argument field is *[+−=][aAcCdDeijPsStTu]* . The prefix + (the default one) causes the attribute(s) to be added; − causes the attribute(s) to be removed; = causes the attributes to be set exactly as the following letters. The letters "aAcCdDeijPsStTu" select the new attributes for the files, see **chattr**(1) for further information.

Passing only = as argument resets all the file attributes listed above. It has to be pointed out that the = prefix limits itself to the attributes corresponding to the letters listed here. All other attributes will be left untouched. Does not follow symlinks.

H

Recursively set file/directory attributes. Lines of this type accept shell−style globs in place of normal path names. Does not follow symlinks.

a, *a+*

Set POSIX ACLs (access control lists). If suffixed with +, the specified entries will be added to the existing set. **systemd−tmpfiles** will automatically add the required base entries for user and group based on the access mode of the file, unless base entries already exist or are explicitly specified. The mask will be added if not specified explicitly or already present. Lines of this type accept shell−style globs in place of normal path names. This can be useful for allowing additional access to certain files. Does not follow symlinks.

A, *A+*

Same as *a* and *a+*, but recursive. Does not follow symlinks.

If the exclamation mark is used, this line is only safe to execute during boot, and can break a running

system. Lines without the exclamation mark are presumed to be safe to execute at any time, e.g. on package upgrades. **systemd−tmpfiles** will execute line with an exclamation mark only if option **−−boot** is given.

For example:

```
# Make sure these are created by default so that nobody else can
d /tmp/.X11−unix 1777 root root 10d

# Unlink the X11 lock files
r! /tmp/.X[0−9]*−lock
```

The second line in contrast to the first one would break a running system, and will only be executed with **−−boot**.

If the minus sign is used, this line failing to run successfully during create (and only create) will not cause the execution of **systemd−tmpfiles** to return an error.

For example:

```
# Modify sysfs but don't fail if we are in a container with a read−only /proc
w− /proc/sys/vm/swappiness − − − − 10
```

Note that for all line types that result in creation of any kind of file node (i.e. *f*/*F*, *d*/*D*/*v*/*q*/*Q*, *p*, *L*, *c*/*b* and *C*) leading directories are implicitly created if needed, owned by root with an access mode of 0755. In order to create them with different modes or ownership make sure to add appropriate *d* lines.

Path

The file system path specification supports simple specifier expansion, see below. The path (after expansion) must be absolute.

Mode

The file access mode to use when creating this file or directory. If omitted or when set to "−", the default is used: 0755 for directories, 0644 for all other file objects. For *z*, *Z* lines, if omitted or when set to "−", the file access mode will not be modified. This parameter is ignored for *x*, *r*, *R*, *L*, *t*, and *a* lines.

Optionally, if prefixed with "~", the access mode is masked based on the already set access bits for existing file or directories: if the existing file has all executable bits unset, all executable bits are removed from the new access mode, too. Similarly, if all read bits are removed from the old access mode, they will be removed from the new access mode too, and if all write bits are removed, they will be removed from the new access mode too. In addition, the sticky/SUID/SGID bit is removed unless applied to a directory. This functionality is particularly useful in conjunction with *Z*.

User, Group

The user and group to use for this file or directory. This may either be a numeric ID or a user/group name. If omitted or when set to "−", the user and group of the user who invokes **systemd−tmpfiles** is used. For *z* and *Z* lines, when omitted or when set to "−", the file ownership will not be modified. These parameters are ignored for *x*, *r*, *R*, *L*, *t*, and *a* lines.

Age

The date field, when set, is used to decide what files to delete when cleaning. If a file or directory is older than the current time minus the age field, it is deleted. The field format is a series of integers each followed by one of the following suffixes for the respective time units: **s**, **m** or **min**, **h**, **d**, **w**, **ms**, and **us**, meaning seconds, minutes, hours, days, weeks, milliseconds, and microseconds, respectively. Full names of the time units can be used too.

If multiple integers and units are specified, the time values are summed. If an integer is given without a unit, **s** is assumed.

When the age is set to zero, the files are cleaned unconditionally.

The age field only applies to lines starting with *d*, *D*, *e*, *v*, *q*, *Q*, *C*, *x* and *X*. If omitted or set to "−", no automatic clean−up is done.

If the age field starts with a tilde character "~", the clean−up is only applied to files and directories one level inside the directory specified, but not the files and directories immediately inside it.

The age of a file system entry is determined from its last modification timestamp (mtime), its last access timestamp (atime), and (except for directories) its last status change timestamp (ctime). Any of these three (or two) values will prevent cleanup if it is more recent than the current time minus the age field.

Note that while the aging algorithm is run a 'shared' BSD file lock (see **flock**(2)) is taken on each directory the algorithm descends into (and each directory below that, and so on). If the aging algorithm finds a lock is already taken on some directory, it (and everything below it) is skipped. Applications may use this to temporarily exclude certain directory subtrees from the aging algorithm: the applications can take a BSD file lock themselves, and as long as they keep it aging of the directory and everything below it is disabled.

Argument

For *L* lines determines the destination path of the symlink. For *c* and *b*, determines the major/minor of the device node, with major and minor formatted as integers, separated by ":", e.g. "1:3". For *f*, *F*, and *w*, the argument may be used to specify a short string that is written to the file, suffixed by a newline. For *C*, specifies the source file or directory. For *t* and *T*, determines extended attributes to be set. For *a* and *A*, determines ACL attributes to be set. For *h* and *H*, determines the file attributes to set. Ignored for all other lines.

This field can contain specifiers, see below.

SPECIFIERS

Specifiers can be used in the "path" and "argument" fields. An unknown or unresolvable specifier is treated as invalid configuration. The following expansions are understood:

Table 1. Specifiers available

EXAMPLES

Example 1. Create directories with specific mode and ownership

screen(1), needs two directories created at boot with specific modes and ownership:

```
# /usr/lib/tmpfiles.d/screen.conf
d /run/screens  1777 root screen 10d
d /run/uscreens 0755 root screen 10d12h
```

Contents of /run/screens and /run/uscreens will be cleaned up after 10 and 10½ days, respectively.

Example 2. Create a directory with a SMACK attribute

```
D /run/cups − − − −
t /run/cups − − − − security.SMACK64=printing user.attr−with−spaces="foo bar"
```

The directory will be owned by root and have default mode. Its contents are not subject to time based cleanup, but will be obliterated when **systemd−tmpfiles −−remove** runs.

Example 3. Create a directory and prevent its contents from cleanup

abrt(1), needs a directory created at boot with specific mode and ownership and its content should be preserved from the automatic cleanup applied to the contents of /var/tmp:

```
# /usr/lib/tmpfiles.d/tmp.conf
d /var/tmp 1777 root root 30d

# /usr/lib/tmpfiles.d/abrt.conf
d /var/tmp/abrt 0755 abrt abrt −
```

Example 4. Apply clean up during boot and based on time

```
# /usr/lib/tmpfiles.d/dnf.conf
r! /var/cache/dnf/*/*/download_lock.pid
r! /var/cache/dnf/*/*/metadata_lock.pid
r! /var/lib/dnf/rpmdb_lock.pid
e  /var/cache/dnf/ − − − 30d
```

The lock files will be removed during boot. Any files and directories in /var/cache/dnf/ will be removed after they have not been accessed in 30 days.

Example 5. Empty the contents of a cache directory on boot

```
# /usr/lib/tmpfiles.d/krb5rcache.conf
e! /var/cache/krb5rcache − − − 0
```

Any files and subdirectories in /var/cache/krb5rcache/ will be removed on boot. The directory will not be created.

/RUN/ AND /VAR/RUN/

/var/run/ is a deprecated symlink to /run/, and applications should use the latter. **systemd−tmpfiles** will warn if /var/run/ is used.

SEE ALSO

systemd(1), **systemd-tmpfiles**(8), **systemd-delta**(1), **systemd.exec**(5), **attr**(5), **getfattr**(1), **setfattr**(1), **setfacl**(1), **getfacl**(1), **chattr**(1), **btrfs-subvolume**(8), **btrfs-qgroup**(8)