NAME

supermin − Tool for creating and building supermin appliances

SYNOPSIS

```
supermin --prepare -o OUTPUTDIR PACKAGE [PACKAGE ...]

supermin --build -o OUTPUTDIR -f chroot|ext2 INPUT [INPUT ...]
```

EXAMPLE

```
supermin --prepare bash util-linux -o /tmp/supermin.d

cat > init <<EOF
#!/bin/sh
mount -t proc /proc /proc
mount -t sysfs /sys /sys
echo Welcome to supermin
bash -i
EOF

chmod +x init
tar zcf /tmp/supermin.d/init.tar.gz ./init

supermin --build /tmp/supermin.d -f ext2 -o /tmp/appliance.d

qemu-kvm -nodefaults -nographic \
         -kernel /tmp/appliance.d/kernel \
         -initrd /tmp/appliance.d/initrd \
         -hda /tmp/appliance.d/root \
         -serial stdio -append "console=ttyS0 root=/dev/sda"
...
Welcome to supermin
bash-4.3#
```

DESCRIPTION

Supermin is a tool for building supermin appliances. These are tiny appliances (similar to virtual machines), usually around 100KB in size, which get fully instantiated on-the-fly in a fraction of a second when you need to boot one of them.

This program used to be called febootstrap. This manual page documents supermin 5.x which is a complete rewrite and quite different from febootstrap 2.x. If you are looking for the febootstrap 2.x tools, then this is not the right place.

BASIC OPERATION

The supermin tool can be used in two modes, **preparing** a tiny supermin appliance, which is done on a build system. And **building**, which takes the supermin appliance and constructs a full, bootable appliance, which is done on the end user's system.

Supermin does not need to be run as root, and generally *should not* be run as root. It does not affect the host system or the packages installed on the host system.

PREPARE MODE

−−prepare creates the tiny supermin appliance in the given output directory. You give it a list of packages that you want installed, and supermin will automatically find the dependencies. The list of packages has to be installed on the host machine.

For example:

```
supermin --prepare bash coreutils -o supermin.d
```

creates a supermin appliance containing the packages `bash` and `coreutils`. Specifically, it creates

some files in directory *supermin.d*. This directory*is* the supermin appliance. (See ‘ ‘SUPERMIN APPLIANCES’’ below).

It is intended that the *−−prepare* step is done on a central build machine, and the supermin appliance is distributed to end users (which is easy because supermin appliances are so small).

BUILD MODE

−−build (previously a separate program called `supermin-helper`) builds the full appliance from the supermin appliance:

```
supermin --build --format ext2 supermin.d -o appliance.d
```

This will create files called *appliance.d/kernel*, *appliance.d/root* etc, which is the full sized bootable appliance.

It is intended that the *−−build* step is done on the end user’s machine at the last second before the appliance is needed. The packages in the supermin appliance (those specified when the supermin appliance was prepared) must be installed on the end user’s machine.

Build and cache

Typically you want to rebuild the appliance on the end user machine only on demand. Supermin has some extra options to make this easy:

```
supermin --build \
  --if-newer --lock /run/user/`id -u`/supermin.lock \
  --format ext2 supermin.d -o appliance.d
```

If multiple programs run this command in parallel, the instances will wait on the lock file. The full appliance only gets rebuilt if it doesn’t exist or if it is older than the input files and host package database.

Note that the lock file **must not** be stored inside the *−o* directory.

PACKAGES

By ‘‘package’’ we mean the RPM, Debian, (etc.) package, eg. `coreutils`, `perl`.

In all cases supermin can only build a supermin appliance which is identical in distro, version and architecture to the host. It does *not* do cross-builds.

OPTIONS

−−help
Display brief command line usage, and exit.

−−build
Build the full appliance from the supermin appliance. This used to be a separate program called `supermin-helper`.

−−copy-kernel
(*−−build* mode only)

Copy the kernel (and device tree, if created) instead of symlinking to the kernel in */boot*.

This is fractionally slower, but is necessary if you want to change the permissions or SELinux label on the kernel or device tree.

−f FORMAT
−−format FORMAT
(*−−build* mode only)

Select the output format for the full appliance.

There is no default. When using*−−b uild* you must specify the *−−format* option.

Possible formats are:

chroot
A directory tree in the host filesystem.

The filesystem tree is written to *OUTPUTDIR* (ie. the *−o* option).

This is called a `chroot` because you could literally *chroot*(1) into this directory afterwards, although it's a better idea to use a container technology (LXC, etc.).

No kernel or initrd is generated in this mode because it is assumed that you will be running the appliance using the host kernel.

ext2
An ext2 filesystem disk image.

The output kernel is written to *OUTPUTDIR/kernel*, a small initramfs which can mount the appliance to *OUTPUTDIR/initrd*, and the ext2 filesystem image to *OUTPUTDIR/root*. (Where *OUTPUTDIR* is specified by the *−o* option).

The filesystem (*OUTPUTDIR/root*) has a default size of 4 GB (see also the *−−size* option).

−−host−cpu CPU
(*−−build* mode only)

Specify the host CPU (eg. `i686`, `x86_64`). This is used as a substring match when searching for compatible kernels. If not specified, it defaults to the host CPU that supermin was compiled on.

−−if−newer
(*−−build* mode only)

The output directory is checked and it is *not* rebuilt unless it needs to be.

This is done by consulting the dates of the host package database (*/var/lib/rpm* etc), the input supermin files, and the output directory. The operation is only carried out if either the host package database or the input supermin files are newer than the output directory.

See also *−−lock* below.

−−include−packagelist
(*−−build* mode only)

Add a */packagelist* file inside the generated chroot or ext2 filesystem, containing a sorted list of all the packages used to build the appliance.

Mostly useful for debugging, as it makes it easier to find out e.g. which version of a package was copied in the appliance.

−−list−drivers
List the package manager drivers compiled into supermin, and whether the corresponding package manager is detected on the current system.

−−lock LOCKFILE
(*−−build* mode only)

If multiple parallel runs of supermin need to build a full appliance, then you can use the *−−lock* option to ensure they do not stomp on each other.

The lock file is used to provide mutual exclusion so only one instance of supermin will run at a time.

Note that the lock file **must not** be stored inside the output directory.

−o OUTPUTDIR
Select the output directory.

When using *−−prepare*, this is the directory where the supermin appliance will be written. When using *−−build*, this is the directory where the full appliance, kernel etc will be written.

Any previous contents of the output directory are deleted, and a new output directory is created.

The output directory is created (nearly) atomically by constructing a temporary directory called something like *OUTPUTDIR.abc543*, then renaming the old output directory (if present) and deleting it, and then renaming the temporary directory to *OUTPUTDIR*. By combining this option with *--lock* you can ensure that multiple parallel runs of supermin do not conflict with each other.

--packager-config CONFIGFILE
(*--prepare* mode only)

Set the configuration file for the package manager. This allows you to specify alternate software repositories.

For ArchLinux, this sets the pacman configuration file (default */etc/pacman.conf*). See *pacman.conf*(5).

For Yum/RPM distributions, this sets the yum configuration file (default */etc/yum.conf*). See *yum.conf*(5).

--prepare
Prepare the supermin appliance.

--use-installed
(*--prepare* mode only)

If packages are already installed, use the contents (from the local filesystem) instead of downloading them.

Note that this can cause malformed appliances if local files have been changed from what was originally in the package. This is particularly a problem for configuration files.

However this option is useful in some controlled situations: for example when using supermin inside a freshly installed chroot, or if you have no network access during the build.

--size SIZE
(*--build* mode only)

Select the size of the output ext2 filesystem, where the size can be specified using common names such as `32G` (32 gigabytes) etc.

If the size is not specified, a default size of 4 GB is used.

To specify size in bytes, the number must be followed by the lowercase letter *b*, eg: `--size10737418240b`.

-v
--verbose
Enable verbose messages.

You can give this option multiple times to enable even more messages:

-v Debugging of overall stages.

-v -v
Detailed information within each stage.

-v -v -v
Massive amounts of debugging (far too much for normal use, but good if you are trying to diagnose a bug in supermin).

-V
--version
Print the package name and version number, and exit.

SUPERMIN APPLIANCES

Supermin appliances consist of just enough information to be able to build an appliance containing the same operating system (Linux version, distro, release etc) as the host OS. Since the host and appliance

share many common files such as */bin/bash* and */lib/libc.so* there is no reason to ship these files in the appliance. They can simply be read from the host on demand when the appliance is launched. Therefore to save space we just store the names of the packages we want from the host, and copy those in (plus dependencies) at build time.

There are some files which cannot just be copied from the host in this way. These include configuration files which the host admin might have edited. So along with the list of host files, we also store a skeleton base image which contains these files and the outline directory structure.

Therefore the supermin appliance normally consists of at least two control files (*packages* and *base.tar.gz*).

packages
: The list of packages to be copied from the host. Dependencies are resolved automatically.

  The file is plain text, one package name per line.

base.tar
base.tar.gz
: This tar file (which may be compressed) contains the skeleton filesystem. Mostly it contains directories and a few configuration files.

  All paths in the tar file should be relative to the root directory of the appliance.

hostfiles
: Any other files that are to be copied from the host. This is a plain text file with one pathname per line.

  Paths can contain wildcards, which are expanded when the appliance is created, eg:

```
/etc/yum.repos.d/*.repo
```

  would copy all of the **.repo* files into the appliance.

  Each pathname in the file should start with a `/` character.

  Supermin itself does not create hostfiles (although before version5, this was the main mechanism used to create the full appliance). However you may drop one or more of these files into the supermin appliance directory if you want to copy random unpackaged files into the full appliance.

excludefiles
: A list of filenames, directory names, or wildcards prefixed by `-` which are excluded from the final appliance.

  This is rather brutal since it just removes things, potentially breaking packages. However it can be used as a convenient way to minimize the size of the final appliance.

  Supermin itself does not create excludefiles. However you may drop one of more of these files into the supermin appliance directory to stop packaged files from being copied into the full appliance.

Note that the names above are just suggestions. You can use any names you want, as supermin detects the contents of each file when it reconstructs the appliance. You can also have multiple of each type of file.

RECONSTRUCTING THE APPLIANCE

The separate mode `supermin --build` is used to reconstruct an appliance from the supermin appliance files.

This program in fact iterates recursively over the files and directories passed to it. A common layout could look like this:

```
supermin.d/
supermin.d/base.tar.gz
supermin.d/extra.tar.gz
supermin.d/packages
supermin.d/zz-hostfiles
```

In this way extra files can be added to the appliance just by creating another tar file (*extra.tar.gz* in the example above) and dropping it into the directory, and additional host files can be added (*zz-hostfiles* in the

example above). When the appliance is constructed, the extra files will appear in the appliance.

MINIMIZING THE SUPERMIN APPLIANCE

You may want to ‘‘minimize’’ the supermin appliance in order to save time and space when it is instantiated. Typically you might want to remove documentation, info files, man pages and locales.

You can do this by creating an excludefiles that lists files, directories or wildcards that you don’t want to include. They are skipped when the full appliance is built.

```
-/boot/*
-/lib/modules/*
-/usr/share/doc/*
-/usr/share/info/*
-/usr/share/man/*
```

Be careful what you remove because files may be necessary for correct operation of the appliance.

KERNEL AND KERNEL MODULES

Usually the kernel and kernel modules are *not* included in the supermin appliance.

When the full appliance is built, the kernel modules from the host are copied in, and it is booted using the host kernel.

USING A CUSTOM KERNEL AND KERNEL MODULES

Supermin is able to choose the best host kernel available to boot the appliance. However you can override this by setting environment variables (see ‘‘ENVIRONMENT VARIABLES’’ below).

If you build a custom kernel (eg. by compiling Linux from source), then you should do something like this:

```
mkdir /tmp/kmods
make bzImage
make modules
make modules_install INSTALL_MOD_PATH=/tmp/kmods

export SUPERMIN_KERNEL=/path/to/linux.git/arch/x86/boot/bzImage
export SUPERMIN_MODULES=/tmp/kmods/lib/modules/3.xx.yy

supermin --build -f ext2 [etc]
```

ENFORCING AVAILABILITY OF PACKAGES

Supermin builds the appliance by copying in the packages listed in *packages*. For this to work those packages must be available. We usually enforce this by adding requirements (eg. RPM `Requires:` lines) on the package that uses the supermin appliance, so that package cannot be installed without pulling in the dependent packages and thus making sure the packages are installed for supermin to use.

ENVIRONMENT VARIABLES

SUPERMIN_KERNEL

If this environment variable is set, then automatic selection of the kernel is bypassed and this kernel is used.

The environment variable should point to a kernel file, eg. */boot/vmlinuz-3.0.x86_64*

SUPERMIN_MODULES

This specifies the kernel modules directory to use.

The environment variable should point to a module directory, eg. */lib/modules/3.0.x86_64/*

SUPERMIN_KERNEL_VERSION

On non-x86 architectures, you may need to set this environment variable if supermin cannot determine the kernel version of `SUPERMIN_KERNEL` just by looking at the file.

SEE ALSO

<http://people.redhat.com/~rjones/supermin/>, *guestfs* (3), <http://libguestfs.org/>.

AUTHORS

- Richard W.M. Jones <http://people.redhat.com/~rjones/>
- Matthew Booth

COPYRIGHT