vmdebootstrap

Release 1.7+git

Neil Williams

May 16, 2017

Contents

CHAPTER 1

VMDebootstrap

Purpose

vmdebootstrap is a helper to install basic Debian system into virtual disk image. It wraps **debootstrap**. You need to run `vmdebootstrap` as root. If the `--verbose` option is not used, no output will be sent to the command line. If the `--log` option is not used, no output will be sent to any log files either.

To use the image, you probably want to create a virtual machine using your preferred virtualization technology, such as `kvm` or `qemu`. Configure the virtual machine to use the image you've created. Then start the virtual machine and log into it via its console to configure it. The image has an empty root password and will not have networking configured by default. Set the root password before you configure networking.

Synopsis

```
$ sudo vmdebootstrap --image=FILE --size=SIZE [--mirror=URL] [--distribution=NAME]
```

Options

--output=FILE	write output to FILE, instead of standard output
--verbose	report what is going on
--no-verbose	opposite of –verbose
--image=FILE	put created disk image in FILE
--size=SIZE	create a disk image of size SIZE (1000000000) in bytes. Suffixes k,K,M,G,T are supported, see qemu-img(1) for more detail.
--tarball=FILE	tar up the disk's contents in FILE
--mirror=URL	use MIRROR as package source (http://httpredir.debian.org/debian/)

--arch=ARCH architecture to use (amd64) — if using an architecture which the host system cannot execute, ensure the `--foreign` option is also used.

--distribution=NAME release to use (defaults to stable). The release needs to be a valid Debian or Ubuntu release name or codename.

--debootstrapopts=OPTS Supply options and arguments to `debootstrap`, separated by spaces. e.g. `--debootstrapopts="variant=buildd no-check-gpg components=main,contrib"`. See **debootstrap (1)** for more information. This option replaces the `--variant` support in previous versions.

--debootstrap-scripts=DIR set the directory containing debootstrap scripts.

--package=PACKAGE install PACKAGE onto system

--custom-package=DEB install package in DEB file onto system (not from mirror) - all dependencies must be available in the specified distribution.

--no-kernel do not install a linux package

--kernel-package=PACKAGE If `--no-kernel` is not used and the auto-selection of the **linux-image-586** or **linux-image-armmp** or **linux-image-$ARCH** package is not suitable, the kernel PACKAGE name can be specified explicitly.

--enable-dhcp enable DHCP on eth0

--root-password=PASSWORD set root password

--lock-root-password lock root account so they cannot login?

--customize=SCRIPT run SCRIPT after setting up system. If the script does not exist in the current working directory, `/usr/share/vmdebootstrap/examples/` will be checked as a fallback. The script needs to be executable and is passed the root directory of the debootstrap and the image name as the only arguments. Use chroot if you need to execute binaries within the chroot created by debootstrap.

--hostname=HOSTNAME set name to HOSTNAME (debian)

--user=USERSTRING create USER with PASSWORD. The USERSTRING needs to be of the format: USER/PASSSWORD.

--owner=OWNER change the owner of the final image from root to the specified user.

--serial-console configure image to use a serial console (Wheezy only)

--serial-console-command (Wheezy only.) Set the command to manage the serial console which will be appended to `/etc/inittab`. Default is `/sbin/getty \-L ttyS0 115200 vt100`, resulting in a line:

```
"S0:23:respawn:/sbin/getty \-L ttyS0 115200 vt100"
```

--sudo install sudo, and if user is created, add them to sudo group

--bootsize=BOOTSIZE If specified, create a /boot partition of the given size within the image. Debootstrapping will fail if this is too small for the selected kernel package and upgrading such a kernel package is likely to need two or three times the space of the installed kernel.

--boottype=FSTYPE Filesystem to use for the /boot partition. (default ext2)

--bootflag=FLAG Flag to set on the first partition. (default none)

--bootoffset=SIZE Space to leave at start of the image for bootloader

--roottype=FSTYPE Filesystem to use for the / (root) partition. (default ext4)

--part-type=PART-TYPE Partition type to use for this image. (default msdos)

--swap=SWAPSIZE If specified, create a swap partition of the given size within the image. Debootstrapping will fail if this results in a root partition which is too small for the selected packages. The minimum swap space is 256MB as the default memory allocation of QEMU is 128MB. A default 1GB image is not likely to have enough space for a swap partition as well.

--foreign=PATH Path to the binfmt_handler to enable foreign support in debootstrap. e.g. `/usr/bin/qemu-arm-static` Note: foreign debootstraps may take a significant amount of time to complete and debootstrap will retry five times if packages fail to install by default.

--use-uefi Setup image for UEFI boot

--no-use-uefi opposite of –use-uefi

--esp-size=SIZE Size of EFI System Partition - requires use-uefi

--extlinux install extlinux (deprecated: default will change in a future release to use grub)

--no-extlinux Skip installation of extlinux. Needs grub, a customize script or alternative bootloader to make the image bootable. extlinux is deprecated and this will become the default in a future release.

--mbr Run install-mbr (default if extlinux used)

--no-mbr opposite of –mbr

--squash=DIRECTORY Run mksquashfs against the rootfs using xz compression — requires `squashfs-tools` to be installed. The squashfs and other files needed to use the squashfs to make a bootable system will be put into the specified directory. The directory will contain a `filesystem.squashfs` as well as the top level contents of the `boot/` directory. (If using UEFI, the `boot/efi` directory as well.) By default, `mksquashfs` is allowed to use all processors which may result in high load. squashfs can also have issues with large root filesystems. These errors can result in truncated files. This is a known bug in squashfs. `vmdebootstrap` will fail if the squashed filesystem is less than 1MB.

--configure-apt Use the specified mirror and distribution to create a suitable apt source inside the VM. Can be useful if debootstrap fails to create it automatically.

--apt-mirror Use the specified mirror inside the image instead of the mirror used to build the image. This is useful if you have a local mirror to make building the image quicker but the image needs to run even if that mirror is not available. Requires `--configure-apt`

--grub Disable extlinux installation and configure grub2 instead. grub2 will be added to the list of packages to install. update-grub will be called once the debootstrap is complete and grub-install will be called in the image.

--no-acpid Disable installation of acpid if not required, otherwise acpid will be installed if `--foreign` is not used.

--sparse Skip optimizing image for compression and keep a sparse image.

--no-sparse opposite of –sparse

--pkglist Output a list of package names installed inside the image. Useful if you need to track the relevant source packages used inside the image for licence compliance.

--dry-run Do not build, just test that the options are valid.

--no-update-initramfs Skip the call to `update-initramfs` for reasons of speed or practicality.

--convert-qcow2 Convert the final raw image to qcow2 format.

--systemd-networkd Use Predictable Network Interface Names

--no-systemd-networkd Do not use Predictable Network Interface Names using systemd-networkd.

Configuration files and settings

--dump-config write out the entire current configuration

--no-default-configs clear list of configuration files to read

--config=FILE add FILE to config files

Logging

--log=FILE write log entries to FILE (default is to not write log files at all); use "syslog" to log to system log, or "none" to disable logging.

--log-level=LEVEL log at LEVEL, one of debug, info, warning, error, critical, fatal (default: debug).

--log-max=SIZE rotate logs larger than SIZE, zero for never (default: 0)

--log-keep=N keep last N logs (10)

--log-mode=MODE set permissions of new log files to MODE (octal; default 0600)

Performance

--dump-memory-profile=METHOD make memory profiling dumps using METHOD, which is one of: none, simple, meliae, or heapy (default: simple)

--memory-dump-interval=SECONDS make memory profiling dumps at least SECONDS apart

Networking

Wheezy support

The `--enable-networking` option uses the `/etc/network/interfaces.d/` source directory, with the default settings for `lo` and `eth0` being added to `/etc/network/interfaces.d/setup`. Other networking configuration can be specified using a customisation script. Localhost settings would be:

```
auto lo
iface lo inet loopback
```

If `--enable-dhcp` is specified, these settings are also included into `/etc/network/interfaces.d/setup`:

```
auto eth0
iface eth0 inet dhcp
```

In addition, wheezy images do not boot if the roottype is specified as the default of `ext4`, so `vmdebootstrap` will fail if a `--roottype` is not specified or is specified as `ext4`.

Jessie and later

In addition, `systemd` in jessie or later introduces PredictableNetworkInterfaceNames which are enabled using the `systemd-networkd` service. If this option is disabled, traditional interface names (like `eth0`) will be used and the predictable names masked using `udev`. Implementing the mask requires updating the initramfs, so the `--update-initramfs` option must not be disabled.

If DHCP is also enabled, the following configuration is used:

```
/etc/systemd/network/99-dhcp.network
```

`systemd` will use the first available match, so this can be overridden by putting another file into place using the customisation scripts, using a lower sorting filename.

Stretch and later

There is no need to use the `--enable-dhcp` option when using `systemd` for networking with stretch or sid. `systemd-resolved` is enabled instead if `systemd-networkd` is specified. (`--enable-dhcp` would simply add an unused entry to `/etc/network/interfaces` for `eth0`.)

```
[Match]
Name=en*

[Network]
DHCP=yes
```

Bootloaders

Unless the `--no-extlinux` or `--grub` options are specified, the image will use `extlinux` as a boot loader. `bootsize` is not recommended when using `extlinux` — use `grub` instead.

Note: Unlike grub, extlinux support requires the installation of packages outside the image which are used to install the extlinux bootloader inside the image. extlinux support also involves the use of `sync` which can cause issues on systems with multiple filesystems mounted, particularly over a network or when building multiple images simultaneously. Therefore, `extlinux` is **deprecated** in vmdebootstrap. The default will change in a future release and `extlinux` support may be dropped once Stretch is released.

extlinux support issues with ext4

VMs using ext4 may not boot when using extlinux - unless the build is performed on Jessie. Builds using ext2 and ext3 work normally.

Important: This problem depends on the **external** distribution, **not** the distribution you are trying to build. When building on Jessie, `extlinux` succeeds but when building on Stretch or Sid, `extlinux` fails to make a bootable system if the filesystem of that system is **ext4**. ext2 and ext3 work.

Version 1.6 of vmdebootstrap adds a warning but allows the build to proceed (to allow for the bug to be fixed). Sadly, downgrading the version of extlinux to the version in Jessie does not fix the problem when building on stretch or sid. Hence, vmdebootstrap can only output a warning.

See also:

http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=833057

Versions of grub2 in wheezy

Grub2 in wheezy can fail to install in the VM, at which point `vmdebootstrap` will fall back to `extlinux`. It may still be possible to complete the installation of `grub2` after booting the VM as the problem may be related to the need to use loopback devices during the `grub-install` operation. Details of the error will appear in the vmdebootstrap log file, if enabled with the `--log` option.

Note: **grub-legacy** is not supported.

`vmdebootstrap` also supports **EFI**. See *UEFI*.

Use `--use-uefi` to use `grub-efi` instead of `grub-pc`. If the default 5MB is not enough space, use the `--esp-size` option to specify a different size for the EFI partition. Registered firmware is not supported as it would need to be done after boot. If the system you are creating is for more than just a VM or live image, you will likely need a larger ESP, up to 500MB.

UEFI

UEFI support requires Grub and `vmdebootstrap` contains a configuration table of the UEFI components required for supported architectures.

There are issues with running UEFI with QEMU on some architectures and a customisation script is available for amd64:

```
# vmdebootstrap --verbose --image jessie-uefi.img --grub  --use-uefi \
  --customize ./examples/qemu-efi-bochs-drm.sh
```

vmdebootstrap supports UEFI for images and for squashfs but the necessary behaviour is different. With an image, an ESP vfat partition is created. With squashfs, the EFI files will be copied into an efi/ directory in the squashfs output directory instead.

There is EFI firmware available to use with QEMU when testing images built using the UEFI support, but this software is in Debian non-free due to patent concerns. If you choose to install ovmf to test UEFI builds, a secondary change is also needed to symlink the provided OVMF.fd to the file required by QEMU: bios-256k.bin and then tell QEMU about the location of this file with the -L option:

```
$ qemu-system-x86_64 -L /usr/share/ovmf/ -machine accel=kvm \
 -m 4096 -smp 2 -drive format=raw,file=test.img
```

To test the image, also consider using the qemu-wrapper.sh:

```
$ /usr/share/vmdebootstrap/qemu-wrapper.sh jessie-uefi.img amd64 /usr/share/ovmf/
```

UBoot

UBoot needs manual configuration via the customisation hook scripts, typically support requires adding u-boot using --package and then copying or manipulating the relevant u-boot files in the customisation script. Examples are included for beaglebone-black.

Some u-boot examples recommend the use of the lba flag on the boot partition, so use the –bootflag option where relevant.

Installation images and virtual machines

:file:vmdebootstrap is aimed principally at creating virtual machines, not installers or prebuilt installation images. It is possible to create prebuilt installation images for some devices but this depends on the specific device. (A 'prebuilt installation image' is a single image file which can be written to physical media in a single operation and which allows the device to boot directly into a fully installed system — in a similar way to how a virtual machine would behave.)

vmdebootstrap assumes that all operations take place on a local image file or directory, not a physical block device / removable media.

vmdebootstrap is intended to be used with tools like qemu on the command line to launch a new virtual machine. Not all devices have virtualisation support in hardware.

This has implications for u-boot support in some cases. If the device can support reading the bootloader from a known partition, like the beaglebone-black, then vmdebootstrap can provide space for the bootloader and the image will work as a prebuilt installation image. If the device expects that the bootloader exists at a specific offset and therefore requires that the bootloader is written as an image not as a binary which can be copied into an existing partition, vmdebootstrap is unable to include that bootloader image into the virtual machine image.

The beagleboneblack.sh script in the examples/ directory provides a worked example to create a prebuilt installation image. However, the beagleboneblack itself does not support virtualisation in hardware, so is unable to launch a virtual machine. Other devices, like the Cubietruck or Wandboard need u-boot at a predefined offset but can launch a virtual machine using qemu, so the cubietruck and wandboard6q scripts in the examples/ directory relate to building images for virtual machines once the device is already installed and booted into a suitable kernel.

It is possible to wrap vmdebootstrap in such a way as to prepare a physical block device with a bootloader image and then deploy the bootstrap on top. However, this does require physical media to be inserted and removed each time the wrapper is executed. To do this, use the --tarball option instead of the --image option. Then setup the physical media and bootloader image manually, as required for the device, redefine the partitions to make space for the rootfs, create a filesystem on the physical media and unpack the vmdebootstrap tarball onto that filesystem. Once

you have working media, an image can be created using dd to read back from the media to an image file, allowing other media to be written with a single image file.

Example

To create an image for the stable release of Debian:

```
sudo vmdebootstrap --image test.img --size 1G \
   --log test.log --log-level debug --verbose \
   --mirror http://mirror.lan/debian/
```

To run the test image, make sure it is writeable. Use the `--owner` option to set mode 0644 for the specified user or use chmod manually:

```
sudo chmod a+w ./test.img
```

If `--log` is also used, consider using `--log-mode` as well so that the logfile is readable by the owner. By default, the log file permissions are 0o600. The logfile itself will be owned by `root`.

Execute using qemu, e.g. on amd64 using qemu-system-x86_64:

```
qemu-system-x86_64 -drive format=raw,file=./test.img
```

(This loads the image in a new window.) Note the use of `-drive file=<img>,format=raw` which is needed for newer versions of QEMU.

There is a `bin/qemu-wrapper.sh <image> <arch>` script for simple calls where the `--owner` option is used, e.g.:

```
$ /usr/share/vmdebootstrap/qemu-wrapper.sh jessie.img amd64
```

There is EFI firmware available to use with QEMU when testing images built using the UEFI support, but this software is in Debian non-free due to patent concerns. If you choose to install `ovmf` to test UEFI builds, a secondary change is also needed to symlink the provided `OVMF.fd` to the file required by QEMU: `bios-256k.bin` and then tell QEMU about the location of this file with the -L option:

```
$ qemu-system-x86_64 -L /usr/share/ovmf/ -machine accel=kvm \
 -m 4096 -smp 2 -drive format=raw,file=test.img
```

To use the `-nographic` option, ensure that the `--serial-console` option is supplied to `vmdebootstrap` and use `-monitor none` when booting the image with QEMU.

For further examples, including u-boot support for beaglebone-black, see `/usr/share/vmdebootstrap/examples`

Notes

If you get problems with the bootstrap process, run a similar bootstrap call directly and chroot into the directory to investigate the failure. The actual debootstrap call is part of the vmdebootstrap logfile. The debootstrap logfile, if any, will be copied into your current working directory on error.

`debootstrap` will download all the apt archive files into the apt cache and does not remove them before starting the configuration of the packages. This can mean that debootstrap can fail due to a lack of space on the device if the VM size is small. vmdebootstrap cleans up the apt cache once debootstrap has finished but this doesn't help if the package

unpack or configuration steps use up all of the space in the meantime. Avoid this problem by specifying a larger size for the image.

Caution: if you are also using a separate /boot partition in your options to `vmdebootstrap` it may well be the boot partition which needs to be enlarged rather than the entire image.

It is advisable to change the mirror in the example scripts to a mirror closer to your location, particularly if you need to do repeated builds. Use the –apt-mirror option to specify the apt mirror to be used inside the image, after boot.

There are two types of examples for ARM devices available with `vmdebootstrap`: prebuilt installation images (like the beaglebone-black) and virtual machine images (cubietruck and wandboard). ARM devices which do not support hypervisor mode and which also rely on the bootloader being at a specific offset instead of using a normal partition will **not** be supportable by vmdebootstrap. Similarly, devices which support hypervisor will only be supported using virtual machine images, unless the bootloader can be executed from a normal partition.

If the host device has a limited amount of RAM or simply to use a different TMP directory when preparing the filesystems, set the `TMPDIR` or `TEMP` or `TMP` environment variables, in line with the underlying support in the python tempfile module.

Developing

Testing vmdebootstrap from git

`vmdebootstrap` uses `yarn` for the test suite, available in the cmdtest package. YARN is a scenario testing tool. Scenarios are written in mostly human readable language, however, they are not free form text. For more information on YARN see the homepage:

```
$ sudo apt -y install cmdtest
```

All commits must pass at least the fast tests. All merges into master need to pass a full test. All additions of new functionality must add fast and build tests — fast tests for any new options and build tests which exercise the new functionality. Build tests can add checks for particular support on the machine running the test and skip if not found or add new environment settings to selectively run some build tests instead of all.

If no arguments are given, the full test suite will be executed:

```
$ yarns/run-tests
```

Warning: Do not run the full test suite if your connection to a Debian mirror is limited or metered. Each build requires a minimum of 2GB free space in tmpfs. A full test takes at least 10 minutes.

When limiting the run to specific tests, each `--env` option needs to be specified separately:

```
$ sudo yarns/run-tests --env TESTS=build --env MIRROR=http://mirror/debian
```

To run a single test, use the `--run` option to specify the name of the scenario (option can be repeated).

pre-commit

All vmdebootstrap developers need to run the fast tests as a pre-commit hook — any patches which fail this test will be rejected:

```
$ ln -s ../../pre-commit.sh .git/hooks/pre-commit
```

The pre-commit hook just runs the fast tests which do not require `sudo`.

Fast tests

The fast checks validate the handling of incompatible option arguments:

```
$ yarns/run-tests --env TESTS=fast
```

Fast tests typically take a few seconds to run.

Build tests

The slow / build tests build multiple images and use `sudo` — a local mirror is strongly recommended.

```
$ sudo yarns/run-tests --env TESTS=build --env MIRROR=http://mirror/debian
```

If `MIRROR` is not specified, a default mirror of `http://httpredir.debian.org/debian/` will be used.

LAVA tests

There is an example `lava-submit.py` script which can be edited to automatically submit QEMU tests to a specified LAVA instance. The images themselves will use local `file://` URLs and therefore the `lava-dispatcher` needs to be installed locally. Configuring LAVA for these tests is a separate topic — please ask on the vmdebootstrap mailing list.

CHAPTER 2

vmdebootstrap for creation of live images

Role of vmdebootstrap

`vmdebootstrap` is limited to the role of generating the rootfs for the live image - the architecture-specific part. `vmdebootstrap` then copies the kernel files out of the rootfs and runs `mksquashfs`.

The files in the directory specified by the `--squash` option are not themselves sufficient to create a live image. Remaining steps include configuration of grub and EFI, addition of other components (like a menu or Debian Installer) and packaging up into a isohybrid image.

vmdebootstrap features

Architecture support

vmdebootstrap has explicit support for foreign architecture bootstraps using qemu static binformat handling as well as support for Debian releases from wheezy onwards.

- This is **not** intended to provide support for all packages in the Debian archive. Some packages do not install correctly with binfmt handling and `vmdebootstrap` should be run natively when the package list is to include these packages.

Whether to use the binfmt_handler or build natively depends on:

1. the availability of a working default kernel for the images built for that architecture and how to configure the bootloader(s) to provide the relevant dtb where needed.
2. the complexity of the package set and compatibility with configuring those packages using qemu-user. Some packages fail if the emulator cannot provide threading support or other mechanisms - package sets with such requirements would need to be built natively. Test with a smaller package set where possible.

live-support package

vmdebootstrap can support adding specific packages but a simpler approach is to use the existing task-* packages and only add packages manually where explicitly needed for a live image, using the `live-support` package.

Running vmdebootstrap for debian-cd

debian-cd runs vmdebootstrap inside a VM in a similar manner to how debian-live currently operates, as both debian-live and vmdebootstrap need to call debootstrap which involves making device nodes and needs to run as root. This outer VM is specific for the release of Debian being built. vmdebootstrap can build older releases and it may be necessary to use a newer version of vmdebootstrap than is present in jessie to build jessie and to use that version to build wheezy.

Remember to use `http://cdbuilder.debian.org/debian/` for the bootstrap operations (–mirror option) and `http://httpredir.debian.org/debian` for the mirror to be used after the image has booted (–apt-mirror option).

Ensure that a user is created (`--user 'user/live'`) and that `sudo` is added to the set of packages to install and the –sudo option is passed to vmdebootstrap to ensure that the user is added to the sudo group. The root user password should also be locked (–lock-root-password).

- Consider using a blank password and enforcing a password to be set upon login for those images which can support this.

`mksquashfs` can fail without indication of why and when it does, the image file can be 4Kb or so of junk. `vmdebootstrap` will fail if the squashfs output is less than 1MB. This can occur if the drive runs out of space but squashfs does not report an error.

Customisation hooks

vmdebootstrap uses a single config file per image type and each config file can have a single customisation script. The config file specifies the architecture of the image and the binformat handler for that architecture (if used), so the customisation hook script can be architecture-specific.

Customisation hook scripts are shell scripts which will be passed a single parameter - the directory which represents the root directory of the final image. These scripts can use standard shell support to include other common functions or call out to utilities known to be installed in the outer VM running vmdebootstrap.

Customisation hooks clearly need to live in a VCS - examples will be carried in the `examples` directory of `vmdebootstrap` and in the `/usr/share/vmdebootstrap/examples` directory. Working scripts based on these examples will likely be within the debian-cd git repo.

Unlike standard vmdebootstrap example scripts, the scripts calling vmdebootstrap itself do not need to use sudo as the call is made inside the outer VM which already has root. Using sudo will work but will output a message: sudo: unable to resolve host JESSIE-debian-live-builder

The building of live images doesn't appear to need changes in the vmdebootstrap package itself. The changes to isolinux to add the menu config, splash screen and to provide access to the install menus can all be done after the generation of the squashfs.

Installing task packages using debootstrap **omits** `Recommended` packages, resulting in a much smaller image which is not expected for a live image. Task selection needs to be done in the customisation hook using the chroot command, at which point the default apt configuration will install the Recommends as well as the Depends packages. Ensure that the image size is big enough.

Use the helpers

`vmdebootstrap` provides helpers for customisation hooks - typically you call a series at the start, do your customisations and call a parallel set before the customisation script finishes. See *Developing live scripts and customisation hooks*.

- *export_env* - When installing using apt in the customisation script, ensure that the debconf non-interactive settings are exported to prevent the install waiting for keyboard interaction:

```
``DEBIAN_FRONTEND=noninteractive``
```

- *mount_proc* - The customisation script needs to mount proc (and possibly other locations like `/sys/`, `/dev/` and `/dev/pts/`) before starting the apt install.
- *cleanup* - cleanup mountpoints at the end of the script.
- Calls to apt should also not output the progress bar but the actual package installation steps should be logged.
- *prepare_apt_source* - Move the image apt sources aside and set the cdimage apt source instead. Use `http://cdbuilder.debian.org/debian/`.
- *replace_apt_source* - At the end of the customisation hook, remove that source and replace the original.
- *disable_daemons* - any daemons installed into the system need to know that the daemon should not be started until boot.
- *remove_daemon_block* - allow installed daemons to start, once all package installations are complete.

CHAPTER 3

Developing live scripts and customisation hooks

`vmdebootstrap` is available in git and in Debian. The live image processing requires several options which are only available in versions of vmdebootstrap newer than version 0.5-2 available in Debian Jessie. vmdebootstrap is able to run on Stretch, Jessie or Wheezy and able to build any suite supported by debootstrap (and and architecture supported by QEMU) on any of those versions of Debian. This leads to a large matrix of build options and hooks.

Calls to vmdebootstrap are best scripted. See the README for notes on which options and settings are required to make a live image using vmdebootstrap.

The 'common' library contains functions and parameters which need to be used in *all* images, including:

```
export_env
mount_proc
disable_daemons
prepare_apt_source

replace_apt_source
remove_daemon_block
cleanup
```

cleanup

Ensure that `proc` is unmounted even if the customisation fails or else the image build itself will fail to unmount *$rootdir*.

export_env

Debconf needs to be set in noninteractive mode to prevent the image build waiting for keyboard intervention.

mount_proc

Many packages require `/proc` to be mounted inside the chroot during installation - cleanup must be specified as a trap if `mount_proc` is used:

```
trap cleanup 0
```

disable_daemons

Packages which include a daemon **must not** start those daemons inside the chroot as this will make the `${rootdir}` appear busy and the unmount will fail. All scripts need to use *remove_daemon_block* after package installation is complete.

prepare_apt_source

The final Debian mirror location is not useful during the build when there is a faster mirror available during the build. This function moves the specified mirror file aside and uses the nearby mirror. Always use with *replace_apt_source*.

Ensure that the mirror and suite are passed as arguments to prepare_apt_source:

```
prepare_apt_source http://mirror/debian jessie
```

remove_daemon_block

After using *disable_daemons*, a policy script remains which needs to be removed to allow daemons to start normally when the image itself is booted. Use `remove_daemon_block` as the next step once package installation is complete.

replace_apt_source

Requires *prepare_apt_source* to have been run first, then undoes the change to the apt sources and cleans up.

TASK_PACKAGES

Some task packages are useful to all images, these are specified here and should be included in the set of packages to be installed using all customisation scripts.

EXTRA_PACKAGES

Packages which are not part of an existing task but which are useful for all images and should be included in the set of packages to be installed using all customisation scripts.

New architectures

The precursor to new architecture support is `vmdebootstrap` support. A default `vmdebootstrap` (with no customisation hook) will need to work and any changes to the settings (e.g. `--no-kernel --package linux-myarch-flavour`) There is default support for some architectures in `vmdebootstrap` (e.g. armhf architectures select the armmp kernel), such support depends on how many users would use the same kernel compared to the number of possible kernel flavours for that architecture.

For a Debian LIVE image, **all** packages must exist in Debian.

The package list also needs a review - some packages will simply not exist for the specified architecture. Some architecture-specific packages need to be added, so each architecture has a particular customisation hook script. Package names frequently change between releases, so the package selection needs to be suite specific as well.

Index