University of Brighton

Project Analysis

Adam Pietrzycki

December 8, 2016

Abstract

In this document I will be keeping a log of my approach and analysis of this project, note that not everything in this will be of any use.

Contents

Chapter 1

Original pi-gen

1.1 /

The build.sh file is the one you run to start generating the Raspbian images. It first sets up a few EXPORTS and SOURCES files from the scripts folder.

1.1.1 config

In the config file you can set an IMG_NAME and an APT_PROXY. The default file can just contain "IMG_NAME='Raspbian'". A quick thing you can do to set up this file is:

```
echo "IMG_NAME='Raspbian'" > config

NOTE:
>  is overwrite if present, create if not.
>> is add to end of file it present, create if not.
```

1.1.2 build.sh

The stages are initially run by this:

```
for STAGE_DIR in ${BASE_DIR}/stage*; do
        run_stage
done
```

For-each loop, runs run_stage method in each directory. This helps us as we can change the main script that is run in each folder to run our integrated Ansible code.

Exports and Source

A bunch of EXPORTS, most important ones are the directories:

```
export BASE_DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")"&&pwd)"
export SCRIPT_DIR="${BASE_DIR}/scripts"
export WORK_DIR="${BASE_DIR}/work/${IMG_DATE}-${IMG_NAME}"
export DEPLOY_DIR="${BASE_DIR}/deploy"
export LOG_FILE="${WORK_DIR}/build.log"
```

This will let us set up static directories when testing the Ansible code, after that can be changed to use dynamic folders.

Stage

This seems to *pushd* the previous stage location to /dev/null which clears it? Then assigns new directories. If CLEAN is set to 1 then it will delete the rootfs folder, I presume this is done during the last stage? If there is a prerun.sh in the current working directory then that will run first. Then it will run run_sub_stage() in each subdirectory of each stage. Finally it unmounts the stage and assigns a few variables; then *popd* into /dev/null?

- `http://ss64.com/bash/pushd.html`
- `http://ss64.com/bash/popd.html`

Sub-Stage

Starts off with *pushd* sub_stage_dir again, then a for loop between 00 and 99. First it pre-seeds the *debconf* and after installs packages-nr using *apt-get*, then packages once packages-nr is done. If there are any patches at this point then this is when they are applied. A few things are done with quilt (to do with patches it seems)??? Coming towards the end of sub-stage now any run.sh and run-chroot.sh files are ran. *Popd* is used again.

- `http://man.he.net/man1/debconf-set-selections`
- `https://linux.die.net/man/8/apt-get`
- `https://linux.die.net/man/1/quilt`

Export-images

1.2 scripts/

1.2.1 common

log

Gets current time and uses a pipe with *tee* to write to the log file.
(`http://man7.org/linux/man-pages/man1/tee.1.html`)

bootstrap

Sets up debootstrap, uses *capsh* to create env I think?
(`http://man7.org/linux/man-pages/man1/capsh.1.html`)

copy_previous

If rootfs folder doesn't exist it will create one, if it does then it uses *rsync* to copy from previous to current stage. This can be avoided to speed things up?
(`http://linuxcommand.org/man_pages/rsync1.html`)

unmount

Does a few checks using $1, unmouts mounted folders using *umount*.
(`http://man7.org/linux/man-pages/man8/umount.8.html`)

unmount_image

First syncs then get *losetup*, then it does a loop through the directories and uses *unmount()*, finally *kpartx* and then *losetup* again.

- `http://linuxcommand.org/man_pages/losetup8.html`
- `http://www.dsm.fordham.edu/cgi-bin/man-cgi.pl?topic=kpartx&ampsect=8`

onchroot

Mounts with bind, uses *realpath and capsh again*:

- $ROOTFS_DIR/proc
- $ROOTFS_DIR/dev
- $ROOTFS_DIR/dev/pts
- $ROOTFS_DIR/sys

(`http://man7.org/linux/man-pages/man3/realpath.3.html`)

update_issue

Prints pi-gen version? This is strange; look into it but it does not look like a priority.

1.2.2 dependencies_check

Dependencies_check, checks if each required tool is installed on the system, the list of packages required can be found in the root directory in ../DEPENDS file.

1.3 export-image/

1.4 export-noobs/

1.5 stage0/

1.6 stage1/

1.7 stage2/

1.8 stage3/

1.9 stage4/

Chapter 2

Ansible pi-gen

Chapter 3

NB

3.1 Kernel panic when Virtualizing

Different versions of pi-gen would fail at different times, normally in stage 4. This was due to code being reverted thus removing the setting a max build stage functionality. The bento boxes are the safest to use though need to install a few more packages, they come with a 50GB virtual disk where as other Vagrant images came with the standard of 8GB and it was a pain to increase.