Ubuntu OEM Recovery framework

Mario Limonciello
Mario_Limonciello@Dell.com

June 4, 2008

Contents

1 Introduction

A common issue for OEMs wanting to ship Ubuntu on their machines is coming up with a sustainable method for creating recovery disks for their customers. The problem is amplified by attempting to make these recovery disks emulate the process used for creating the factory installation in the first place. Multiply this by a large number of variable hardware and software configurations offered at system creation time and the problem is then unmaintainable.

The logical solution is to create a framework that can then be reused and adapted to work with a variety of teams without having a single team's work interfering with or detrimentally affecting another's workflow.

2 Framework overview

This framework provides a solution that allows OEMs to provide hooks into each portion of the installation. These hooks can be workarounds, drivers for hardware enablement, software, documentation, and localization. The exact same hooks are then replicated on the resultant system allowing users to create a recovery disk with all of the same benefits.

2.1 Basic layout

The framework itself is laid out in the following structure:

- ./grub

- ./docs

- ./preseed

- ./scripts

- ./scripts/chroot-scripts

- ./scripts/chroot-scripts/os-pre

- ./scripts/chroot-scripts/fish

- ./scripts/chroot-scripts/os-post

- ./debs

- ./debs/main

- ./isolinux

This is intended to supplement the existing directory structure of an Ubuntu DVD. Each of these directories (and the scripts in them) will be explained as they are brought up.

3   Factory Installation

When a standard Ubuntu installation is performed, the contents of the DVD
are copied onto a system with a few minor modifications from the hardware
detected during installation as well as input parameters related to localization.
For this factory installation, the same basic install is performed as a standard
Ubuntu installation, but hooks have been added to do things before install
begins, modify input variables during install, and to add on additional changes
to the resultant system.

3.1   Preparing the installation

To get started, checkout a copy of the framework and download a copy of the
latest Ubuntu DVD. You will need a blank hard drive to work with as well.

1. First set up the partition structure on the hard drive. It is expected that
   you have 2 small partitions with the rest of the drive unallocated. The
   first partition is intended for diagnostic utilities and should be 50-100 mb.
   The second partition is your recovery partition. You should set it to be 5
   GB and either fat32 or ext3.

2. Download an Ubuntu DVD and extract it's contents to the recovery par-
   tition. Be sure to extract the hidden .disk directory too (this is critical).

3. Download a copy of the OEM framework and extract it "on top" of the
   extract recovery partition. There are a few files that need to be overwrit-
   ten.

4. You will now need to prepare the drive to boot onto this second partition.
   This can be achieved using a DOS filesystem and linld, a linux filesystem
   and kexec, or by installing GRUB to the drive. Do whatever is most con-
   ducive to your environment. You will need to use this kernel command
   line to get started:
   ```
   kernel /casper/vmlinuz boot=casper edd=on preseed/file=/cdrom/preseed/oem.seed
   noninteractive noprompt
   ```

3.2   Booting the installation

The installation begins by passing a kernel command line to the kernel shipped
with the Ubuntu DVD. This command line will generally look something like
this:

```
kernel /casper/vmlinuz boot=casper edd=on preseed/file=/cdrom/preseed/oem.seed
noninteractive noprompt
```

Let's break up that command line to explain what each portion is doing.

Option	Purpose
kernel	The option to menu.lst to indicate we are loading a kernel.
/casper/vmlinuz	The kernel image we are loading.
boot=casper	Indicates that this is a live disk booting and to use unionfs.
edd=on	Turns on Enhanced Disk Drive support, very useful in multi disk boxes.
preseed..	The seed file we are passing to the Ubuntu installer.
noninteractive	Indicates that we are using the non-GUI installer.
noprompt	Tells the system to not prompt at the end of install, but just reboot.

The non GUI installer was selected in case the event comes up that video hardware is not supported in X until after postinstall scripts are complete. If you are sure that video hardware is supported by X in all scenarios, you can replace **noninteractive** with **automatic-ubiquity**.

A more complete summary of kernel options available during and related to installation is available at `https://wiki.ubuntu.com/DesktopCDOptions`

3.3   Preboot phase

As soon as the kernel loads, casper will load the contents of the oem.seed file. It will use this to determine what the `early_command` is. This early command starts immediately after the kernel loads but before all of the init scripts start. The current early command will call `scripts/bootstrap.sh`.
This script will then load the OEM framework environment, `environ.sh` and then individually launch all scripts listed in scripts/bootstrap. If any command in any of the scripts returns anything but zero, `scripts/bootstrap/FAIL-SCRIPT` will be called. You can choose to do anything you like in this script. If all of the scripts complete successfully, `scripts/bootstrap/SUCCESS-SCRIPT` will be called.

`05-CONFIRM-REINSTALL.sh` checks if this is a reinstallation, and will not be used for Factory Installation. It checks for the existence of the **REINSTALL** kernel parameter to determine if this is a reinstallation.

`06-CONFIRM-DVD.sh` checks if this is a reinstallation from DVD, and will not be used for Factory Installation. It checks for the existence of the **DVDBOOT** kernel parameter to determine if this is a DVD being booted.

`10-format.sh` formats the disks to start. This ensures that the installation will always look the same when the actual Ubuntu install process gets started. Doing this in all cases (Factory, Reinstall from HD, Reinstall from DVD) will ensure a successful and standard installation.

3.4 Install phase

After the `early_command` completes, the actual Ubuntu installation will begin.
All of the data that was read in oem.seed will be applied to the normal input
parameters for the Ubuntu installer, Ubiquity. The exact contents of this file will
be explained later. If the installation is successful, the `success_command` from
oem.seed is ran. If the installation failed, the `failure_command` from oem.seed
is launched.

3.5 Post-install phase

As soon as the installation finishes, the postinstallation script is launched in
both success and failure. If the installation failed for any reason, the fail script
`scripts/chroot-scripts/FAIL-SCRIPT` will be launched. In this script you
can perform cleanup or analyze what failed in more detail.

In the event of success, this script will prepare the post installation environ-
ment similar to the preinstall script did. The difference here is that all of the
scripts launched in postinstall are launched on the resultant system. This means
that you can perform package installation, modify configuration files, and tweak
anything as necessary in the environment. If any script in here fails with a return
code greater than 0, `scripts/chroot-scripts/FAIL-SCRIPT` will get launched.
The post-installation is broken into three subphases.

3.5.1 os-pre

The *os-pre* phase is the first of the three phases. It is intended to handle files
and functions that if interfered with by hardware or software enablement teams
at inappropriate times would break.

Currently two scripts are contained in here:

> `10-grub.sh` will add a menu option to the GRUB menu.lst for choosing a
> hard drive based recovery. If another team decided that another options
> needs to be added to menu.lst, this ensures that the OS requirement is
> placed first.

> `80-install-debs.sh` will install debian archives that provide dependen-
> cies for other steps as well as archives that don't need to be installed in a
> particular order.

Again, this section is intended to be used by an OS development team for
things related to the OS requirements. Hardware and software enablement will
happen in the other phases.

3.5.2 fish

The *fish* phase is the second of the three phases. It is intended to handle hardware and software enablement. Teams that need to modify configuration files for default hardware behavior or add drivers should do so here. Also, teams that need to add extra software that didn't fall in the *os-pre* step (possibly due to dependency resolution issues) can do so here. The framework ships with this directory empty, but some examples are available for things that can go in here.

3.5.3 os-post

The *os-post* phase is the last of the three phases. It is intended for final cleanup steps. The OS team can include scripts here that will be OS wide workarounds as well as final cleanup steps that need to happen to prepare the seal on the box.

Shipped here are 4 starting scripts.

`50-handle-docs.sh` is intended to allow documentation and translation teams to include important documents with the base installation. To prevent having to provide a high bar for them learning how to package these, this script will simply copy over any relevant documentation onto the installation in a standard location.

`60-fix-apt-sources.sh` is used to add additional vendor specific APT sources.

`90-oem-prepare.sh` is used to put the machine into OEM mode for the first boot. This causes it to ask for a user name and password on the first boot.

`99-active-partition.sh` sets the active bootable partition on the system. This intentionally happens as a last step because of the possibility of anything not completing or failing earlier due to a power outage. The system will boot up to the recovery partition and the entire process can start over if this step isn't ran.

4 Recovery installation

During installation, grub is installed to the recovery partition and a default menu.lst is shipped with the OEM framework. If during boot, a user selects the option to reinstall their system, GRUB will chainload to the second partition. This menu.lst is presented to the user giving them the option to recover. The entire process will proceed just like a factory install with the exception of an added boot parameter, **REINSTALL**.

Earlier we skipped over a script, `05-CONFIRM-REINSTALL.sh`. This script checks for the extra boot parameter, and presents the user with a dialog making sure

that they really want to wipe the disk. This script comes before `10-format.sh`. If they don't agree to the dialog, the machine is rebooted preventing any data loss.

5   DVD installation

Within the framework, a directory isolinux/ is shipped that provides an extra menu option to the standard Ubuntu DVD boot menu. Similar to the HD based recovery, the DVD based recovery provides an extra kernel command line parameter, **REINSTALL**. DVD boots also provides the kernel command line parameter **DVDBOOT** indicating that it is being ran from a DVD.

When `06-CONFIRM-DVD.sh` is ran, some extra behavior then happens to copy the contents of the MBR and the recovery DVD to the user's system. The rest of the installation then proceeds like normal, but instead off of the DVD. This is a little bit slower, but just as functional.
DVD creation will be explained later on.

6   Use case examples

In the examples directory you will find a handful of real life examples that have been used with this framework. They have been used for hardware enablement when not all the changes could be easily represented in a debian archive.
If you have a use case that you have found this particularly useful for, please submit it back to this project. Adding it to the examples section will benefit all parties.

6.1   Hardware enablement

There are two exampless of hardware enablements:

> **70-nvidia-driver.py** installs an NVIDIA proprietary graphics driver, removes the warning from Jockey, and then activates it in the xorg.conf. The debs for the NVIDIA graphics driver are not included on the Ubuntu DVD, so they are shipped with this script.

> **71-thinkfinger.py** will install and activate thinkfinger support. The thinkfinger debs come with the Ubuntu DVD so they can be installed from there. The script also turns on the PAM section to allow swiping to login.

6.2   Software enablement

There are also two examples of software enablement:

50-install-DKMS.sh sets up the DKMS package. These debs don't come on the DVD so they are provided independently. Arguably this fits better in os-pre because some hardware enablement relies on it.

90-add-lbm.sh add the linux-backports-modules package to the system. This package provides updated drivers and firmware for some cards.

6.3 Bug workarounds

85-no-ehci.sh shows how to work around a bug that came with the OS that prevents proper functionality. In the event that it's too late or complex to fix a bug directly in the OS, this is a good example of how to script a workaround into the system. It modifies just the files necessary and in a fashion that would not cause harm later on when packages get updated.

7 Debugging

The most important thing about debugging is identifying which phase of the installation needs to be debugged. The methodology used for each can vary depending on the problem you are trying to solve.

7.1 Boot problems

If you have an underlying kernel problem, you will likely get a panic directly upon boot. If you don't, you may be dropped down to a console at which point you can look at `casper.log` in the current directory, or the output of `dmesg` for more information. If you are discovering problems during this phase, it's important to file bugs on these issues early as they are most critical and may require a significant amount of time to fix.

7.2 Preboot phase

If the preboot phase is failing, you will be dropped to a console. Debugging this can be done by adding echo statements to the scripts and looking at the output of casper.log. Alternatively, you can handle debugging directly in your `FAIL-SCRIPT`.

7.3 Install phase

Debugging the install phase is likely to be the most difficult because the problems can be either in the implementation of the preseed file or a bug directly in the installer. The installer will always place debug output in `/var/log/syslog` and possibly `/var/log/installer/debug` depending on the type of crash you encounter.

A useful debug strategy is to boot up a live disk and attempt to simulate

an installation environment. You can perform all of the steps normally done in your preboot phase manually. After doing this, you can load your preseed file:

```
cat oem.seed | sudo debconf-set-selections
```

Once the preseed file is loaded, you can launch Ubiquity with either the automatic-ubiquity or noninteractive frontends in addition to **debug** mode.

```
ubiquity -d noninteractive or ubiquity -d --automatic
```

The logging will be much more verbose in `/var/log/installer/debug`.

Be sure that if you have any bugs about the installer to file to include your oem.seed in the bug and any of the installation logs that you can provide.

7.4 Post Install phase

The post install phase would be most likely to have problems due to unknown interactions of different team's post installation scripts. Debugging can be performed interactively or after a failed installation by looking at logs. The post install phase will log all activity to `/var/log/installer/chroot.sh.log` on the resultant system.

You can also interactively prevent each script from launching one by one by creating a file `/tmp/superhalt.flg` on the live filesystem. This will create a flag, `/tmp/halt.flg` before each script is ran. When you want the next script to launch, just remove `/tmp/halt.flg`. If you would like all of the scripts to continue on, remove both `/tmp/superhalt.flg` and `/tmp/halt.flg`

8 Preseeding

Each of the items in the preseed file provided with the framework is documented, but if you would like to learn more in depth you can take a look at the Ubuntu preseeding guide at `https://help.ubuntu.com/8.04/installation-guide/i386/appendix-preseed.html`.

9 DVD Creation

For DVD creation, the current framework expects that you will burn a bootable DVD that loads the standard isolinux setup shipped in the framework.

Additionally, you will have to provide a copy of the system's MBR in a file titled `mbr.bin` in the root of the DVD image and a gzipped copy of the utility partition in a file titled `upimg.bin` in the root of the DVD image. You can choose to remove these portions of the framework as you see fit.

An example application for creating these types of DVDs is available in the Ubuntu archive. Take a look at the project entitled **dell-recovery**.