Timo's Rescue CD Set: A Do-it-Yourself Bootable Debian Rescue CD

Rescue me!

Creating your own customized, bootable rescue CD is by no means a trivial task. Fortunately the toolset introduced in this article allows even relatively inexperienced users to achieve presentable results and so be ready for the day when a rescue CD is the only option to save the data on a computer. BY TIMO BENK

Most Linux users will be familiar with the problem: you compile a new kernel, for some reason or other, copy the kernel to the */boot* directory, and while your machine is going down, it suddenly occurs to you that it might have been better to run lilo *before* typing *halt*. Bad luck.

You start rummaging through your desk. That boot diskette you created way back when, while installing your system, must be there somewhere. Pity the kernel on the disk doesn't support your SCSI interface, and of course that's what your Linux hard disk happens to be attached to. Murphy's law strikes once more.

This, or a similar situation was what prompted this project some two years ago. Originally, the idea was just to produce a boot diskette of my own to avoid this scenario in future. Unfortunately. a floppy just was not big enough to store all those useful and desirable tools, so it had to be a bootable CD.

This quickly led to two scripts that were downloadable from *sourceforge.net*; "Who knows, maybe someone else can use them, I thought". In the meantime these two scripts have developed into a complete toolset that allows anyone to create their own boot CD in next to no time.

The CD is based on the Debian GNU/Linux distribution, and the scripts have all been tested in this environment. However, major issues are unlikely with most other distributions.

System Configuration

After downloading the archive from [1], you can type *tar xpjf timos_rescue_cd_set-0.9.8.tar.bz2* to store the toolset in a directory called *timos_rescue_cd_set-0.9.8*. It is important to run the tar command with the *root* UID, to give privileges for creating device files otherwise.

One aim of the toolset is simple handling and customization. This is why the directory structure on the CD is freely accessible below the *_system/_cd* directory, where it can be customized to reflect your requirements.

If your requirements are not too exotic, you will not need to make any changes to the System. The *config_system* file contains parameters for many basic configurations.

The *system_config* script is used to transfer them. This is where you configure the network environment of login password for example.

The next step is to customize the *config_build* file. The most important parameters here are *BASE* and *SCSIDEV* which specify where to install the toolset and which CD drive to use as a CD burner. Our tip: Try the *cdrecord -scanbus* command to display the available drives and their SCSI addresses. If you will not be using a CDRW medium, you can comment out the *SCSIBLANK* parameter, as the burn session will otherwise terminate with an error message. That covers the most important parameters, additional modifications will follow later.

Package Management

The *apt* tool provided with Debian GNU/Linux is an excellent package manager. One of *apt*'s most important features is the ability to download and install packages directly off the Internet. As the rescue system is based on Debian, packages can be installed, removed, or updated quite comfortably.

To allow users without expert knowledge of Debian to start selecting packages immediately, the *utils* subdirectory contains a few useful scripts that are used as apt wrappers. Note that these scripts can only be run in the *utils* directory. If at all possible, you should always use this

Listing 1: Kernel Options

```
01 # CONFIG_BLK_DEV_IDECD is not
   set
02 CONFIG_BLK_DEV_IDESCSI=y
03 CONFIG_BLK_DEV_RAM=y
04 CONFIG_BLK_DEV_INITRD=y
05 CONFIG_DEVFS_FS=y
06 CONFIG_TMPFS=y
07 CONFIG_ISO9660_FS=y
08 CONFIG_ZISOFS=y
09 CONFIG_RAMFS=y
10 CONFIG_CRAMFS=y
11 CONFIG_BLK_DEV_LOOP=y
12 CONFIG_BLK_DEV_SR=y
```

method to customize the system. This will ensure that a nearly original Debian GNU/Linux is placed on your CD, the advantage being that your rescue system will be based on a mature distribution. Any packages you add will thus be interoperable.

The *pkg_install* script installs new packages to the system. To add the compact *joe* editor to your rescue CD, simply launch the following command in the *utils* directory.

```
# ./pkg_install joe
```

If you need to remove a package, the required tool is *pkg_remove*.

pkg_list provides an overview of the packages available for installing, and *pkg_show_installed* displays a list of installed packages.

The last, but possibly most important, tool in the pkg-set is *pkg_update*. It updates the package descriptions and installs any updates that are available. You should run *pkg_update* before creating a CD to resolve bugs or remove vulnerabilities. Debian users will probably just smile on hearing that updating across release boundaries is possible.

Creating the CD

That completes the system configuration; now it is time to create your first CD using the *build* and *system_config* to do so. *system_config* transfers the values from the *config_system* to the *_system/_cd* tree, and *build* finally creates the CD itself. Another point of interest is the fact that the build script calls the *mkcramfs* amongst others. This tool is stored in the *cramfsprogs* package on a Debian system. If the tool is not already installed, you will need to install

Boot loaders à la carte

The *BOOTLOADER* parameter in *config_build* allows you to select from a whole range of boot loaders. Each one of them has its specific advantages and disadvantages. Table 1 provides an overview.

Apart from isolinux, all of these boot loaders boot from a boot disk image, and use the El Torito specification from 1994, which supports 2.88 MByte images. Unfortunately, not every BIOS that you will come across today actually supports this standard; thus the 2.88 MByte images will not necessarily boot on every system and you may need to take this into account beforehand.

advertisement

Table 1: Advantages and disadvantages of various boot loaders

BootLoader	Advantage	Disadvantage
Syslinux	Floppy based boot disk images, boots on almost any system. If you want the CD to run on as many computers as possible, you should opt for this, along with a 1.44 Mbyte boot disk image.	Syslinux should not be used on any filesystem that extends past cylinder 1024.
Grub	Looks good, has a command shell that allows you to boot unconfigured Systems.	It requires slightly more space than Syslinux.
Lilo	Lilo is everybody's darling.	We have heard of issues with lilo in connection with El Torito boot disk images.
Isolinux	The only boot loader that is not based on the El Torito specification. No size restrictions on the kernel image and initial ramdisk.	Isolinux will not boot on all systems.

INFO

[1] Timo's Rescue CD Set: *http://rescuecd.sourceforge.net*

[2] The rescue CD in business card format: *http://rescuecd.sourceforge.net/README.bbc*

[3] VMware: *http://www.vmware.com/*

[4] Zisofs Tools: *ftp://ftp.kernel.org/pub/linux/utils/fs/zisofs/RPMS/i386/*

[5] Roll your own Linux Rescue or Setup CD: *http://www.phenix.bnl.gov/~purschke/RescueCD/*

[6] Cloop: *http://www.knopper.net/download/knoppix/*

it before creating a CD. Now insert the CD in the writer and you are ready to go.

```
# ./system_config
# ./build
```

If everything went as planned, the new CD should be in your CD writer.

If you do not like to leave home without Linux, some manual steps will allow you to create a rescue CD in business card format. Refer to [2] for a guide.

Advanced config_build

There are a number of editable parameters in the *config_build* file. We will be taking a closer look at *ISOIMAGEONLY*, *COMPRESSEDFS* and *BOOTLOADER* in the following sections.

The *ISOIMAGEONLY* parameter specifies whether to burn a CD or simply create an ISO9660 image. The image is stored as *rescuecdiso* in the root directory and can be used by VMware [3] as a CD ROM drive. You do not need to reboot to test the system, simply reset VMware.

The *BOOTLOADER* parameter allows you to choose between syslinux, isolinux, grub and lilo. Each boot loader has its own specific advantages and disadvantages; it is difficult to say which one is best suited. The "Boot loaders à la Carte" box provides an overview. Isolinux is the easiest to handle, and this is preferable if you know that your hardware will not pose any problems.

Bootloader other than isolinux need a floppy image. The BIOS uses the El Torito specification to access the CD similar to a floppy drive. The image size can be either 1.44 or 2.88 MBytes, and is specified using the *FLOPPYSIZE* parameter. The advantage of a 2.88 MByte floppy image is obvious: it provides twice the amount of space for the kernel, and thus allows the kernel to support more hardware.

As is often the case, however, there is also a downside; not all BIOSes support the El Torito specification, and thus 2.88 MByte floppy disk images will not boot on every system.

Concocting a Kernel

To avoid this, you can of course compose your own kernel, but it will need to fulfill one or two conditions. One of them is hard coded support for SCSI CD ROM drives, the IDE SCSI emulation, the initial ramdisk, loop devices and the following filesystems *devs*, *ramfs*, *tmpfs*, *cramfs*, and ISO9660 including the *zisofs* option. On the other hand, hard coded IDE CD ROM support is to be avoided to ensure that the IDE SCSI emulation is available for all CD ROM drives.

Listing 1 shows the relevant entries in the */usr/src/linux/.config* file.

After compiling the new kernel, you need to copy it to the *_system/_kernel* directory. The *vmlinuz* symlink in the same directory needs to be modified to point to the new kernel.

The *_system/_kernel/modules/2.4.20* is replaced by the current modules and the *modules.** files are moved to the *_system/_cd/var/files* directory. Make symlinks in the *_system/_kernel/modules/2.4.20* directory to allow the next CD to boot its own kernel. ■

Compressing the CD

There are several reasons for compressing the data on the rescue CD. For one thing, this will allow you to copy the whole CD to memory. And this means that the CD ROM drive is free for writing operations. However, you do need a correspondingly large amount of RAM. Compressing the CD reduces memory usage by about a factor of 2, allowing you to launch the default system and store it in the memory on a computer with 128 Mbytes of RAM. Over the years various compression methods have been added to the toolset; each one of them has its own special advantages and disadvantages.

zisofs was the most logical approach and can be specified using the *COMPRESSEDFS* parameter in *config_build*. This ISO9660 filesystem extension is supported by newer kernel versions. Unfortunately, this requires a patched mkisofs version [4]; the version supplied with most distributions does not support compressed files. Zisofs compresses the files at filesystem level and this does mean some overhead. Thus, zisofs returns the worst compression factor.

cramfs, which is specified by the *CRAMFS* parameter in the *config_build* file returns better results in this area. Cramfs also compresses at filesystem level, but causes less overhead. The storage space gained by compression does cause some disadvantages: the maximum file size drops to 16 MBytes and the maximum size for the whole filesystem is 256 MBytes. User and group IDs are restricted to 8 bits.

The last method to be introduced was the *cloop* module [6]. The corresponding parameter is *CLOOP*. The module compresses the filesystem at block level and is used much like a loop device, as the name suggests. This method produced the best compression factor for filesystems, however, the *cloop* module needs to be re-compiled each time you exchange the kernel.