Linux Complete Backup and Recovery HOWTO

Charles Curley

charlescurley at charlescurley dot com

Revision History

Revision 2.15d Revised by: c^2
Development for 2.15. Added new script make.gdisk. Other, minor, changes. All changes are now shown in the change log
Revision 2.14 2022-04-08 Revised by: c^2
Fdisk now returns the string gpt in either upper or lower case. Other improvements.
Revision 2.13 2019-07-24 Revised by: c^2
save.metadata no longer changes all /dev/sda device names to /dev/hda names. Major changes in the README.
Revision 2.12 2018-06-09 Revised by: c^2
Added checksum generation and testing to improve reliability.
Revision 2.11 2017-07-29 Revised by: c^2
Several improvements after working with Debian 9.
Revision 2.10 2017-07-12 Revised by: c^2
We now check to see if the Perl UUID module is installed and in Perl's module path. Also, this is our first release under git.
Revision 2.9 2014-01-06 Revised by: c^2
Added a new file, which is a list of excludes for use by get.target.
Revision 2.8 2011-06-13 Revised by: c^2
Maintenance release. Many edits due to changes in technology. URL fixes.
Revision 2.7 2010-01-12 Revised by: c^2
More improvements in make.fdisk: Removed a bug related to UUIDs and found a better way to get them.
Revision 2.6 2009-12-24 Revised by: c^2
Improvements in make.fdisk: we now have a better way to get UUIDs, and we can preserve them for swap space.
Revision 2.5 2008-04-24 Revised by: c^2
Changed shebangs in the backup scripts to point to bash instead of sh because some ubunutu installations don't make sh
Revision 2.4 2007-11-08 Revised by: c^2
Some Debian/Ubuntu adaptations. see the notes below on them.
Revision 2.3 2007-05-26 Revised by: c^2
Changes for FHS compliance. Changes in `save.metadata` to handle the libata problem.
Revision 2.2 2006-07-11 Revised by: c^2
Clarified that the ZIP disk is not required, and there are alternatives.
Revision 2.1 2006-03-28 Revised by: c^2
Added notes for NTFS. Edited the To Do list. Started work on LVM and using Finnix (http://www.finnix.org/).
Revision 2.0 2005-10-12 Revised by: c^2
Notes for Fedora Core 4. Removed notes for older versions of FC and Red Hat. Also, changes in the writeup and scripts to
Revision 1.8 2005-02-19 Revised by: c^2
Added notes for Fedora Core 3
Revision 1.7 2004-05-11 Revised by: c^2
Adjusted copyright language.
Revision 1.6 2004-04-29 Revised by: c^2
Added Knoppix (http://www.knoppix.org/) notes, Syslinux, PPART, QtParted, some other rescue CDs, and made some fixes
Revision 1.5 2003-12-19 Revised by: c^2
Fedora 1 and GRUB notes.
Revision 1.4 2003-08-17 Revised by: c^2

Abstract

Imagine your disk drive has just become a very expensive hockey puck. Imagine you have had a fire, and your computer case now looks like something Salvador Dalĩ would like to paint. Now what?

Total restore, sometimes called bare metal recovery, is the process of rebuilding a computer after a catastrophic failure. In order to make a total restoration, you must have complete backups, not only of your file system, but of partition information and other data. This HOWTO is a step-by-step tutorial on how to back up a Linux computer so as to be able to make a bare metal recovery, and how to make that bare metal recovery. It includes some related scripts.

Some notes on burning CD-ROMs, and more on files to exclude.
Revision 1.3 2003-04-24 Revised by: c^2
Substituted new email address and URL for old.
Revision 1.2 2003-02-12 Revised by: c^2
Added Red Hat 8.0 notes, support for FAT32, split the first stage restore scripts, and other minor changes. Notes on Amand
Revision 1.1 2002-09-10 Revised by: c^2
New code to handle ext3 partitions in `make.fdisk`, and a note on `initrd`.
Revision 1.0 2002-07-24 Revised by: c^2
We now use bz2 compression in the first stage, have the run time option to check for bad blocks, and have a script that run

Table of Contents

1. Introduction **5**
1.1. Copyright Information 5
1.2. Disclaimers 5
1.3. New Versions 5
1.4. Credits 5
1.5. Feedback 6
1.6. Translations 6
2. Overview **6**
2.1. Limitations 7
3. Preparation **8**
3.1. Installing the ZIP Drive 9
3.2. Backup Server 9
4. Creating the Stage 1 Back Up **9**
4.1. The Archive 11
4.2. Theme And Variations 11
4.2.1. No ZIP drive 12
4.2.2. CD-ROM 12
4.2.3. NFS 13
4.2.4. Multiple ZIP disks 13
4.2.5. Excluding From First Stage Saving 13
4.2.6. Initrd 13
5. First Stage Restore **14**
5.1. Booting 14

5.1.1. Finnix 14
5.1.2. Knoppix 14
5.1.3. tomsrtbt 14
5.2. Restoration 15
5.2.1. Automated 15
5.2.2. Manually 16
5.2.3. Finishing Touches 16
6. Second Stage Restoration 17
7. Distribution Specific Notes 19
7.1. Debian/Ubuntu 19
7.2. Fedora 19
7.3. Knoppix 19
7.4. Finnix 20
7.5. Grml 20
8. Application Specific Notes 20
8.1. Logical Volume Manager 20
8.2. LUKS 20
8.3. systemd 21
8.4. Selinux 21
8.5. Tripwire 21
8.6. Squid 21
8.7. Arkeia 21
8.8. Amanda 22
8.9. NTFS 22
8.10. UEFI 23
9. Some Advice for Disaster Recovery 23
10. What Now? 23
10.1. To Do 23
11. The Scripts 24
11.1. First Stage 24
11.1.1. `make.fdisk` 24
11.1.2. `make.gdisk` 48
11.1.3. `make.dev.hda` 65
11.1.4. `make.lvs` 66
11.1.5. `mount.dev.hda` 68
11.1.6. `mount.lvs` 69
11.1.7. `dev.hda` 70
11.1.8. `dev.hda.sfd` 70
11.1.9. `save.metadata` 71
11.1.10. `restore.metadata` 80
11.1.11. `first.stage` 83
11.2. Second Stage 85
11.2.1. `back.up.all` 85
11.2.2. `back.up.all.ssh` 86
11.2.3. `restore.all` 87
11.2.4. `restore.all.ssh` 88

11.3. Backup Server Scripts 88
11.3.1. `get` 89
11.3.2. `post.sh` 90
11.3.3. `get.target` 91
11.3.4. `restore` 93
11.4. Miscellaneous Files 94
11.4.1. `install` 94
11.4.2. `ssh.sh` 96
12. Resources 97
A. GNU Free Documentation License 99
0. PREAMBLE 99
1. APPLICABILITY AND DEFINITIONS 100
2. VERBATIM COPYING 101
3. COPYING IN QUANTITY 101
4. MODIFICATIONS 101
5. COMBINING DOCUMENTS 103
6. COLLECTIONS OF DOCUMENTS 103
7. AGGREGATION WITH INDEPENDENT WORKS 103
8. TRANSLATION 103
9. TERMINATION 104
10. FUTURE REVISIONS OF THIS LICENSE 104
11. How to use this License for your documents 104

1. Introduction

The normal bare metal restoration process is: install the operating system from the product disks. Install the backup software, so you can restore your data. Restore your data. Then you get to restore functionality by verifying your configuration files, permissions, etc.

The process and scripts explained in this HOWTO will save re-installing the operating system. The process explained here will restore only files that were backed up from the production computer. Your configuration will be intact when you restore the system, which should save you hours of verifying configurations and data.

1.1. Copyright Information

Copyright © 2001 through last date of modification Charles Curley and distributed under the terms of the GNU Free Documentation License (GFDL) license, stated below. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, with no Front-Cover Texts, and with no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have any questions, please contact <linux-howto at metalab.unc.edu>.

1.2. Disclaimers

No liability for the contents of this documents can be accepted by the author, the Linux Documentation Project (http://www.tldp.org/) (LDP) or anyone else. Use the concepts, examples and other content at your own risk. There may be errors and inaccuracies that may damage your system. Proceed with caution, and, although errors are unlikely, the author take no responsibility for them.

All copyrights are held by their by their respective owners, unless specifically noted otherwise. Use of a term in this document should not be regarded as affecting the validity of any trademark or service mark.

Naming of particular products or brands should not be seen as endorsements.

You are strongly recommended to take a backup of your system before major installation and backups at regular intervals. In addition, you are strongly recommended to use a sacrificial experimental computer when mucking with the material, especially the scripts, in this HOWTO. When I first wrote this, virtual machines were a rarity. Now they are readily available.

1.3. New Versions

You can find this document at its home page (https://charlescurley.com/Linux-Complete-Backup-and-Recovery-HOWTO.php) or at the Linux Documentation Project (http://www.tldp.org/) web site in many formats. Please comment to <charlescurley at charlescurley dot com>

Depending on your browser, you may have to hold down the shift button while you click on these in order to get them to download.

1.4. Credits

This document is derived from two articles originally published in *Linux Journal* (http://www.linuxjournal.com/). My thanks to *Linux Journal* for reverting the rights to those articles, thereby helping make this HOWTO possible.

Thanks to Joy Y. Goodreau for excellent HOWTO editing, and to David Palomares for correcting the spelling of Salvador Dalī's name.

Also, thanks to Pasi Oja-Nisula (mailto:pon at iki dot fi) for a bug fix and information on Knoppix (http://www.knoppix.org/).

1.5. Feedback

Feedback is most certainly welcome for this document. Without your corrections, suggestions and other input, this document wouldn't exist. Please send your additions, comments and criticisms to me at: <`charlescurley at charlescurley dot com`>.

1.6. Translations

Not everyone speaks English. Volunteers are welcome.

2. Overview

The process shown below is not easy, and can be hazardous to your data. Practice it before you need it! Do as I did, and *practice on a sacrificial computer*! Virtual machines are excellent for the purpose.

The original target computer for this HOWTO was a Pentium computer. Originally, it had a Red Hat (http://www.redhat.com) 7.1 Linux server or workstation installation on one IDE hard drive. Since then, I have used a number of computers, and they have been ugraded to Red Hat 8.0 and Fedora Cores 1, 3 and 4. (http://fedora.redhat.com/). The target computer does not have vast amounts of data because the computer was set up as a "sacrificial" test bed. That is, I did not want to test this process with a production computer and production data. Also, I did a fresh installation before I started the testing so that I could always re-install if I needed to revert to a known configuration.

> **NOTE:** The sample commands will show, in most cases, what I had to type to recover the target system. You may have to use similar commands, but with different parameters. It is up to you to be sure you duplicate your setup, and not the test computer's setup.

The basic procedure is set out in W. Curtis Preston, *Unix Backup & Recovery* (http://www.oreilly.com/catalog/unixbr/), O'Reilly & Associates, 1999, which I have favorably reviewed in *Linux Journal* (http://m.linuxjournal.com/article/3839). However, the book is a bit thin on specific, real-time questions. For example, exactly which files do you back up? What metadata should you preserve, and how? This document explores those questions.

Before beginning the process set forth in this HOWTO you will need to back up your system with a typical backup tool such as Amanda, BRU™, tar, Arkeia® or cpio. The question, then, is how to get from toasted hardware to the point where you can run the restoration tool that will restore your data.

Users of RPM based Linux distributions (Fedora (http://fedora.redhat.com/), e.g.) should also save RPM metadata as part of their normal backups. The following is in one of the scripts in this HOWTO:

```
bash# rpm -Va | sort +2 -t ' ' | uniq > /etc/rpmVa.txt
```

It provides a basis for comparison after a bare metal restoration.

For Debian systems, the optional program **debsums** provides an analogous facility.

To get to this point, you must have:

- Your hardware up and running again, with replacement components as needed. The BIOS should be correctly configured, including time and date, and hard drive parameters. At the moment, there is no provision for using a different hard drive.
- When I started this project, I used a ZIP® drive (http://www.iomega.com/zip/products/par100_250.html). Now, they are rather cramped for space and can be inconvenient. You can substitute a USB flash disk, NFS mount, CD-RW or other medium. Just be sure that the Linux distribution you use for first stage restore supports your medium. For historical reasons, this document will refer to the ZIP drive (http://www.iomega.com/zip/products/par100_250.html); please substitute the medium of your choice. There is more discussion of alternatives below in the section on Theme And Variations.
- Your normal backup media: tape hard drive, etc.
- A minimal Linux system that will allow you to run the restoration software, which we will call the restoration Linux.

To get there, you need at least two stages of backup, and possibly three. Exactly what you back up and in which stage you back it up is determined by your restoration process. For example, if you are restoring a tape server, you may not need networking during the restoration process. So only back up networking in your regular backups.

You will restore in stages as well. In stage one, we build partitions, file systems, etc. and restore a minimum of files from the ZIP disk. The goal of stage one is to be able to boot to a running computer with a network connection, tape drives, restoration software, or whatever we need for stage two.

The second stage, if it is necessary, consists of restoring backup software and any relevant databases. For example, suppose you use Arkeia and you are building a bare metal recovery ZIP disk for your backup server. Arkeia keeps a database on the server's hard drives. You can recover the database from the tapes, if you want. Instead, why not tar and gzip the whole arkeia directory (at /usr/knox), and save that to another computer over NFS or SSH? Stage one, as we have defined it below, does not include X, so you will have some experimenting to do if you wish to back up X as well as your backup program. Some restore programs require X.

Of course, if you are using some other backup program, you may have some detective work to do to. You will have to find out the directories and files it needs to run. If you use tar, gzip, cpio, mt or dd for your backup and recovery tools, they will be saved to and restored from our ZIP disk as part of the stage one process describe below.

The last stage is a total restoration from tape or other media. After you have done that last stage, you should be able to boot to a fully restored and operational system.

2.1. Limitations

This HOWTO is restricted to making a minimal backup such that, having then restored that backup to new hardware ("bare metal"), you can then use your regular backups to restore a completely working system. This HOWTO does not deal with your regular backups at all.

Even within that narrow brief, this HOWTO is not exhaustive. You still have some research, script editing, and testing to do.

The scripts here restore the partition data exactly as found on the source hard drive. This is nice if you are restoring on an identical computer or at least an identical hard drive, but that is often not the case. For now, there are two remedies (which will make more sense after you've read the rest of the HOWTO):

- Edit the partition table input file. I've done that a few times. You can also do this to add new partitions or delete existing ones (but edit the scripts that use the partition table input file as well).
- Hand build a new partition table and go from there. That is one reason why `restore.metadata` does not call the hard drive rebuilding script. Use the rebuilding script.

The scripts shown here only handle ext2/3/4, FAT12, FAT16 and FAT32. Until some eager volunteer supplies code for doing so in these scripts, you will need other tools for backing up and restoring file systems we haven't covered. However, see the note below on NTFS. Partition Image (http://www.partimage.org/) looks like a useful candidate here.

3. Preparation

> **WARNING:** Do your normal backups on their regular schedule. This HOWTO is useless if you don't do that.

Build yourself a restoration Linux disk. For this I now use Finnix (http://www.finnix.org/). Grml (https://grml.org/) also works nicely. See the notes below. I have used Knoppix (http://www.knoppix.org/) in the past. See the notes on Knoppix below. However, everything here is command line. We don't need a GUI. A GUI-less distribution (such as Finnix) will boot faster and can load itself into memory (so you can use the CD drive) even on a minimal machine.

In the past, I have used tomsrtbt (http://www.toms.net/rb/). It is well documented and packs a lot of useful tools onto one floppy diskette. Unfortunately, the changes I've had to make in the scripts to handle more recent Linux systems cause problems for tomsrtbt (http://www.toms.net/rb/). The tomsrtbt (http://www.toms.net/rb/) 2.0.103 tar is based on busybox (http://www.busybox.net/), so remarks about it may apply to other Linux disties which use busybox. And tomsrtbt has other problems.

> **WARNING:** The version of **tar** included in tomsrtbt (http://www.toms.net/rb/) does not preserve ownership when it restores. This may cause problems for applications like Amanda. A backup and restoration tool, Amanda has several directories owned by its own user. The solution is:
>
> - Note which directories and files are not owned by root.
> - Note their owners.
> - Arrange to set the ownership correctly as part of the restoration process. E.g:
>
>   ```
>   bash# chown -R amanda:disk /var/lib/amanda
>   ```
>
>   You can also add that line to your scripts for second state restoration, such as `restore`.

> **WARNING:** tomsrtbt (http://www.toms.net/rb/) does not support restoring owners by UID/GID. To make backups suitable for restoring with tomsrtbt (http://www.toms.net/rb/), remove the tar command line option "--numeric-owner" from the command line options for tar in the function crunch in the script `save.metadata`.

Next, figure out how to do the operating system backup you will need so that you can restore your normal backup. I used to follow Preston's advice and use an Iomega parallel port ZIP drive. The drives get approximately 90 MB of useful storage to a disk. I need about 85 MB to back up my desktop, so a 100MB ZIP drive may be pushing your luck. These days I use CD-RWs, NFS or USB sticks. For more on those, see the sections on using CD-ROMs and NFS.

3.1. Installing the ZIP Drive

Installing the ZIP drive is covered in the ZIP Drive HOWTO (http://www.tldp.org/HOWTO/mini/ZIP-Drive.html), available at the Linux Documentation Project (http://www.tldp.org/) and at its home page, http://www.njtcom.com/dansie/zip-drive.html.

3.2. Backup Server

You can set up a backup server for this process. Scripts on the backup server interact with the target machines (including itself) via SSH. They assume that your backup server user (root here, for simplicity) can log in with no password to the targets. This is necessary for unattended backups.

First, create a suitable directory to keep all the backups in. We'll call it `backs`. In backs, create a directory for each target computer. The first field in the directory should be the host name. Subsequent fields can be other useful information. For example, to preserve the last backup of a target before an installation of a new version of the distribution, I use an abbreviation for the distribution, e.g. "fc5". Fields are separated with periods ("."). So, for example, `tester.f7`. The host name is required because the scripts use that to determine which host to back up.

Copy the scripts `get` and `restore` into each target's directory. Then customize them for each host as needed.

Also create in `backs` a directory called `scripts` and put in it the script `get.target`. This is a library for the backup and restore scripts. It performs actions common to all backups and restorations.

4. Creating the Stage 1 Back Up

Having made your production backups, you need to preserve your partition information so that you can rebuild your partitions.

The scripts `make.fdisk` and `make.gdisk` scan a hard drive for partition information, and saves it in several files. The first is an executable script, one per hard drive, called (for example) `make.dev.x` (where "x" is the name of the device file, e.g. hda, sda, or nvme). Second is `mount.dev.x`, which creates mount points and mounts the newly created partitions on them. The next, `dev.x`, is the commands necessary for **fdisk** to build the partitions. Last is an input file, e.g. `dev.sda.sfd`, for **sfdisk** to create partions. (**sfdisk** is preferable and the scripts will used it if they find it.) You specify which hard drive you want to build scripts for (and thus the file names) by naming the associated device file as the argument to `make.fdisk`. For example, on a typical IDE system,

```
bash# make.fdisk /dev/hda
```

spits out the scripts `make.dev.hda`, `mount.dev.hda` and the input files for **fdisk** and **sfdisk**, `dev.hda` and `dev.hda.sfd`, respectively.

In addition, if `make.fdisk` encounters a FAT partition, it preserves the partition's boot sector in a file named `dev.xy`, where x is the drive's device name (e.g. sdc, hda) and y is the partition number. The boot sector is the first sector, 512 bytes, of the partition. This sector is restored at the same time the partitions are rebuilt, in the script `make.dev.hda`. However, except for EFI partitions, we make no attempt to lay down a file system in FAT partitions.

Fortunately, the price of hard drives is plummeting almost as fast as the public's trust in politicians after an election. So it is good that the output files are text, and allow hand editing. That's the most difficult but most flexible way to rebuild on a larger replacement drive. (See the To Do list.)

The script `save.metadata` preserves other metadata as needed. The script saves the partition information in the file `fdisk.hda` in the root of the ZIP disk. It is a good idea to print this file and your `/etc/fstab` so that you have hard copy should you ever have to restore the partition data manually. You can save a tree by toggling between two virtual consoles, running **fdisk** in one and catting `/etc/fstab` or `/fdisk.hda` as needed. However, doing so is error prone.

You will also want to preserve files relevant to your restoration method. For example, if you use NFS to save your data, you will need to preserve `hosts.allow`, `hosts.deny`, `exports`, etc. Also, if you are using any network-backed restoration process, such as Amanda or Quick Restore, you will need to preserve networking files like HOSTNAME, hosts, etc. and the relevant software tree. The simplest way to handle these and similar questions is to preserve the entire `/etc` directory.

There is no way a 100 MB ZIP drive is going to hold a server installation of a modern distribution of Linux. We have to be much more selective than simply preserving the whole kazoo. What files do we need?

- The boot directory.
- The `/etc` directory and subdirectories.
- Directories needed at boot time.
- Device files in `/dev` unless you are using udev.

To determine the directories needed at boot, we look at the boot initialization file `/etc/rc.sysinit` (or equivalent). It sets its own path like so:

```
PATH=/bin:/sbin:/usr/bin:/usr/sbin
export PATH
```

Trial and error indicated that we needed some other directories as well.

In reading the script `save.metadata`, note that we aren't necessarily saving files that are called with absolute paths.

We may require several iterations of back up, test the bare metal restore, re-install from CD and try again, before we have a working backup script. While I worked on this HOWTO, I made five such iterations before I had a successful restoration. That is one reason why it is essential to use scripts whenever possible. Test thoroughly!

One thing you can do on an RPM based system is use the **rpm** program to determine which files are where. For example, to get a complete list of the files used by the openssh package, run:

```
bash# rpm -ql openssh
```

Similarly of debian boxes, using **apt-file**.

```
bash# apt-file search openssh-server
```

There are some things you don't need, like the man pages. You can inspect each one and decide whether to back it up or not.

> **WARNING:** The second stage of restoration is run without overwriting previously restored files. This means that the files restored in the first stage are the ones that will be used after full restoration. So update your bare metal backups whenever you update files in these directories!

> **WARNING:** Recent kernels have incorporated a new ATA (IDE) hard drive driver, libata. Because of this, parallel ATA drives (PATA) now show up as SCSI drives, as serial ATA (SATA) have always done. However, not all rescue distributions use this new driver. There is a line toward the bottom of `save.metadata` which very carefully replaces "/dev/sda" with "/dev/hda". Use this as a template if you have multiple IDE hard drives. Comment it out or delete it if this is not an issue for you.
>
> Note that there is no guaranteed mapping! Systems with multiple hard drives may have confusing mappings. Be sure to edit this line carefully. Check it if you add or remove a hard drive of any interface type to or from your system!
>
> N.B: if you have libata IDE drive issues, the grub-install line at the end of `restore.metadata` won't work. If it doesn't, use your rescue disk to do the same. Or burn and boot to the boot image that is made as part of the first stage backup. Boot to it and do the second stage restore as usual. The second stage restore should re-run `grub-install` or you can run it manually.

4.1. The Archive

All of this gets stored into an archive under `/var/lib/bare.metal.recovery`. Each day a first stage backup is made a new directory is prepared, with the date encoded as YYYYMMDD, and the day's archive deposited therein. It is up to you to prune obsolete archives. It is a good idea to keep at least one old archive around in case the computer crashes while you are making an archive. If a second archive is made in a day, the earlier one for that day is replaced.

The files in the archive directory include a `README.txt`, which has information about the backup and the computer the backup was made on. Other files are there in case hand intervention is required.

Below the daily archive directory are several text files and three directories. The scripts reside in `bin`, the tarballs in `data`, and information about the system such as partitions and LVM volume backups are in `metadata`.

To create a CD, simply use a script or graphical tool to create a CD starting at the daily archive directory. It is up to you to be sure your archive will fit onto your medium, or to make other arrangements.

Also in the archive are checksum files for each daily archive. Use those to check the integrity of the daily archives. This is useful for checking archives when you copy them to other computers (such as a backup server) or to the target prior to resoration. If you launch the bare metal backup from the backup server using `get`, that script will copy the checksum files to the server, and check the archive copy on the server.

4.2. Theme And Variations

4.2.1. No ZIP drive

This backup process used to require you to have the ZIP disk drive present at each backup. It now creates the archive in a directory, which you can back up over the net. Then you only need to build a ZIP disk (with **cp -rp**) on the backup server when you need to restore.

The backup process will be faster than directly writing to the ZIP drive, but you should check that the resulting directory will fit on your ZIP disk (with the output of **du -hs $target.zip** in the script `save.metadata`)! See the definition of the variable `zip` in that script.

One of my laptops has problems running both a network card and a ZIP drive, so this is the process I use to back it up. I keep a backup image as well as the current one, so that I have a fallback in case the computer crashes during a backup.

4.2.2. CD-ROM

This is similar to the no ZIP drive option above. Save your backups to a directory on your hard drive, as noted. Then use **mkisofs** to create an ISO 9660 image from that directory, and burn it. This does not work with some CD-ROM based restoration Linuxes, like Knoppix (http://www.knoppix.org/), because the Linux has to have the CD-ROM drive. Unless you have two CD-ROM drives, say one in a USB clamshell. I have a DVD burner set up this way with exactly this in mind. Better, have Finnix (http://www.finnix.org/) load itself into memory on boot and then use the CD-ROM drive from which you booted.

These remarks should also apply to DVDs.

Test your CDs on the drive you will use at restoration time. If you find you need to hack the scripts, you can copy them to `/tmp`, usually a RAM drive, and edit them there. The scripts will run there. As a RAM disk is volatile, be sure to save your changes before you reboot!

Much of rest of this section is old information. I recommend you use Finnix (http://www.finnix.org/).

You can remastering (http://www.knoppix.net/wiki/Knoppix_Remastering_Howto) Knoppix or Finnix (http://www.finnix.org/) with your first and second stage backups on the CD-ROM. You should also be able to remaster Finnix (http://www.finnix.org/Remastering_Finnix).

These days many computers come with a CD-ROM drive but no floppy diskette. And floppy drives do fail. So it's a good idea to burn your CD-ROM with a bootable image on it. The bad news is that the "El Torito" format supports 1.2 MB, 1.44 MB and 2.88 MB floppy images, and tomsrtbt (http://www.toms.net/rb/) uses a 1.7 MB floppy. The good news is that you can get a 2.88 MB version, `tomsrtbt-2.0.103.ElTorito.288.img`, from the same mirrors where you get the floppy image. Place a *copy* [1] in the root directory of the backup files. Then use the **mkisofs** command line option -b to specify `tomsrtbt-2.0.103.ElTorito.288.img` as the boot image file.

The only down side of this process is that many older BIOSes do not support 2.88 MB floppy images on CD-ROMs. Most of those will boot to a tomsrtbt (http://www.toms.net/rb/) floppy.

An alternative is to use Syslinux (http://syslinux.zytor.com/). It is not dependent on a floppy diskette image, and you can build your own CD with a number of tools, such as tomsrtbt (http://www.toms.net/rb/), on it.

You may have to adjust the BIOS options to allow the computer to boot to CD-ROM drive. If you can't do that, either because the BIOS won't support booting to CD-ROM, or because you can't get into the BIOS, see Smart Boot Manager (SBM) (http://sourceforge.net/projects/btmgr/) as described in the Resources.

One variant is to dispense with the tarballs in the first stage, and create a tarball of the entire system. When you build your restoration CD, put the monster tarball in the data directory of the CD. The scripts will pick that up and restore for you, combining the first and second stages. This eliminates a lot of the cruft related to permissions and ownership in `restore.metadata` and `save.metadata`

4.2.3. NFS

If you back up across your network to a backup server, you will have all the files on it you need. Set up the directory where you keep all your backups as an NFS export.

Then, on Finnix (http://www.finnix.org/), do the following (tab completion is very nice here):

```
# mkdir /mnt/nfs
# /etc/init.d/portmap start
# mount server:/path/of/exportedfs /mnt/nfs
# cd /mnt/nfs/.../bin
```

Now restore as usual.

There are several advantages to NFS for this job: You don't have to worry about space on a CD-ROM or ZIP drive (http://www.iomega.com/zip/products/par100_250.html). You can edit scripts on the server and they are preserved when you reboot the target.

4.2.4. Multiple ZIP disks

By splitting up the two first stage scripts, `restore.metadata` and `save.metadata`, you could spread the first stage metadata across multiple ZIP disks.

4.2.5. Excluding From First Stage Saving

There are time when you need to squeeze a few megabytes from the first stage data, especially when you are pushing the limit of your ZIP disk. The function **crunch** in the script `save.metadata` takes multiple parameters to feed to **tar**. It can also take the **--exclude** parameter. So, for example, you can exclude the `samba` and `X11` directories under `/etc` like so:

```
crunch etc --exclude etc/samba --exclude etc/X11 etc
```

Why those two? Because they're hard drive space hogs and we don't need them when booting after the first stage.

If you keep multiple kernels around, you can eliminate the modules for all of the kernels you won't boot to. Check your `lilo.conf` or `/boot/grub/menu.lst` to see which kernel you will use, and then check `/lib/modules` for module directories you can exclude.

How to find more good candidates for exclusion? List the target directories with **ls -alSr** for individual files, and **du | sort -n** for directories.

Another, neater, way to exclude directories is to put a complete list of directories into a file, then refer to it via the tar option `--exclude-from=FILENAME`.

4.2.6. Initrd

If your system uses an initial RAM disk, or initrd, to boot, make sure that `restore.metadata` creates the directory `/initrd`. The easiest way to do this is to ensure that it is included in the list of directories used in the directory creating loop toward the end.

Your system will probably use an initrd if it boots from a SCSI drive or has root on an ext3fs partition. Check `/etc/lilo.conf` or `/boot/grub/menu.lst` to see if it calls for one.

When in doubt, check the root directory to see if `/initrd` is there.

5. First Stage Restore

5.1. Booting

The first thing to do is to verify that the hardware time is set correctly. Use the BIOS setup for this. How close to exact you have to set the time depends on your applications. For restoration, within a few minutes of exact time should be accurate enough. This will allow time-critical events to pick up where they left off when you finally launch the restored system.

If you want to log in to the restoration OS, via SSH, see the script `ssh.sh`.

5.1.1. Finnix

One option for booting Finnix (http://www.finnix.org/) is the "toram" option, which lets you move the whole kazoo into RAM. That in turn should let you load another CD, with your first stage data, into the CD drive.

Should you run into problems with **grub-install**, see Restoring bootloaders (https://www.finnix.org/Restoring_bootloaders) in the Finnix wiki.

If it is possible that there is a LUKS or LVM partition on the hard drive, but you don't want to access them on boot (because you are about to write over them anyway, say), add the **nocrypt** or **nolvm** options to the boot command line. There are more boot parameters (https://www.finnix.org/Boot_parameters) available.

5.1.2. Knoppix

These instructions will probably work with other CD-ROM or USB pen Linuxes, but you may have to vary them to suit.

Before booting Knoppix (http://www.knoppix.org/), make sure your ZIP drive (or substitute) is installed on a parallel port, either `/dev/lp0` or `/dev/lp1`. Knoppix does not load the parallel port ZIP drive driver for you. Instead, use the command **modprobe ppa** (as root) to install it.

Boot Knoppix (http://www.knoppix.org/) as usual. I find it faster and more useful to boot to a console. At the boot menu, use the command "knoppix 2". Then become the root user, with **su -**. For the password, just hit return.

5.1.3. tomsrtbt

Before booting tomsrtbt (http://www.toms.net/rb/), make sure your ZIP drive is installed on a parallel port, either `/dev/lp0` or `/dev/lp1`. The start-up software will load the parallel port ZIP drive driver for you.

The next step is to set the video mode. I usually like to see as much on the screen as I can. When the option to select a video mode comes, I use mode 6, 80 columns by 60 lines. Your hardware may or may not be able to handle high resolutions like that, so experiment with it.

5.2. Restoration

These instructions assume you are running tomsrtbt (http://www.toms.net/rb/). If you are using a different Linux for your restore system, you may have to adjust these instructions a bit. For example, you should always run these scripts as root even if some other user gives you the requisite privileges.

Once the restoration Linux has booted and you have a console, mount the ZIP drive. It is probably a good idea to mount it read only. On tomsrtbt (http://www.toms.net/rb/):

```
# mount /dev/sda1 /mnt -o ro
```

Check to be sure it is there:

```
# ls -l /mnt
```

On Knoppix (http://www.knoppix.org/) or Finnix (http://www.finnix.org/), you may want to make a directory under `/mnt` and mount it there, like so:

```
# mkdir /mnt/zip
# mount /dev/sda1 /mnt/zip -o ro
```

Now cd into the mounted device, and into the `bin` directory below it. On Finnix (http://www.finnix.org/), for example:

```
# cd /mnt/zip/bin
```

The scripts assume you are in this directory, and call data files relative to it. At this point, you can run the restoration automatically or manually. Use the automated restore if you don't need to make any changes as you go along.

One consideration here is whether you have multiple hard drives. If your Linux installation mounts partitions on multiple hard drives, you must mount the root partition first. This is to ensure that mount point directories are created on the partition where they belong. The script first.stage will run the scripts to mount the drives in the order in which they are created. If you have created them (in the script `save.metadata`) in the order in which they cascade from root, the mounting process should work just fine.

If you have multiple hard drives, and they cross-mount, you are on your own. Either combine and edit the scripts to mount them in the correct order, or do it manually.

5.2.1. Automated

The automatic process calls each of the manual scripts in proper order. It does not allow for manual intervention, say for creating file systems that this HOWTO does not support. To run the first stage restore automatically, enter the command:

```
# first.stage
```

If you want to check for back blocks, add the **-c** option.

5.2.2. Manually

Run the script(s) that will restore the partition information and create file systems. You may run them in any order, so long as they build dependencies in the correct order. You can read the script first.stage to get an idea of the order. e.g.:

```
# ./make.dev.hda
```

If you want to check for back blocks, add the **-c** option.

This script will:

- Clean out the first 1024 bytes of the hard drive, killing off any existing partition table and master boot record (MBR).
- Recreate the non-LVM partitions from the information gathered when you ran `make.fdisk`.
- Make ext3+ file systems on non-LVM partitions and Linux swap partitions as appropriate. If you provide the **-c** option to the script, it will also check for bad blocks.
- Make some types of FAT partitions.

Now is a good time to check the geometry of the drive. Sometimes different versions of Linux pick up different geometries, so the geometry implicit in the file `dev.hdX` is incorrect. To force it to be correct on Knoppix (http://www.knoppix.org/), edit `make.dev.x`. Use the -C, -H and -S options to fdisk to specify the cylnders, heads and sectors, respectively. Those you can get from the file `fdisk.hdX`. Then re-run it.

> **NOTE:** If you have other operating systems or file systems to restore, now is a good time to do so. When you've done that, reboot to your restoration Linux and continue your first stage restoration.

If you have LVM volumes to restore, now is the time to run `make.lvs` and `mount.lvs`.

Now run the script(s) that create mount points and mount the partitions to them.

```
# ./mount.dev.hda
```

Once you have created all your directories and mounted partitions to them, you can run the script `restore.metadata`.

```
# ./restore.metadata
```

This will restore the contents of the ZIP drive to the hard drive to give you a minimal bootable system.

You should see a directory of the ZIP disk's root directory, then a list of the archive files as they are restored. Tar on tomsrtbt (http://www.toms.net/rb/) will tell you that tar's block size is 20, and that's fine. You can ignore it. Be sure that lilo prints out its results:

```
Added linux *
```

That will be followed by the output from a "**df -m**" command.

5.2.3. Finishing Touches

If you normally boot directly to X, you could have some problems.

On System V installations, to be safe, the first stage script changes the run level in `/target/etc/inittab` to 3. Note: different distributions use different run level schemes. 3 works on Red Hat derived distributions; it may not on others.

On systemd installations (and probably others), you can avoid problems by booting into recovery mode. Or edit the command line at boot time (https://fedoraproject.org/wiki/Systemd#Boot_Kernel_Command_Line). If there is a better way to do this, I'd like to hear about it.

You can now gracefully reboot. Remove the medium from your boot drive if you haven't already done so, and give the computer the three fingered salute, or its equivalent:

```
# shutdown -r now
```

or

```
# reboot
```

or

```
# systemctl reboot
```

The computer will shut down and reboot.

6. Second Stage Restoration

As the computer reboots, go back to the BIOS and verify that the clock is more or less correct.

Once you have verified the clock is correct, exit the BIOS and reboot to the hard drive. You can simply let the computer boot in its normal sequence. You will see a lot of error messages, mostly along the lines of "I can't find blah! Waahhh!" If you have done your homework correctly up until now, those error messages won't matter. You don't need linuxconf or apache to do what you need to do.

> **NOTE:** As an alternative, you can boot to single user mode (at the lilo prompt, enter **linux single**), but you will have to configure your network manually and fire up sshd or whatever daemons you need to restore your system. How you do those things is very system specific.

You should be able to log into a root console (no X, no users, sorry). You should now be able to use the network, for example to NFS mount the backup of your system.

If you did the two stage backup I suggested for Arkeia, you can now restore Arkeia's database and executables. You should be able to run

```
/etc/rc.d/init.d/arkeia start
```

and start the server. If you have the GUI installed on another computer with X installed, you should now be able to log in to Arkeia on your tape server, and prepare your restoration.

> **NOTE:** When you restore, read the documentation for your restoration programs carefully. For example, tar does not normally restore certain characteristics of files, like suid bits. File permissions are set by the user's umask. To restore your files exactly as you saved them, use tar's p option. Similarly, make sure your restoration software will restore everything exactly as you saved it.

To restore the test computer:

```
bash# restore.all
```

If you used tar for your backup and restoration, and used the -k (keep old files, don't overwrite) option, you will see a lot of this:

```
tar: usr/sbin/rpcinfo: Could not create file: File exists
tar: usr/sbin/zdump: Could not create file: File exists
tar: usr/sbin/zic: Could not create file: File exists
tar: usr/sbin/ab: Could not create file: File exists
```

This is normal, as tar is refusing to overwrite files you restored during the first stage of restoration.

Then reboot. On the way down, you will see a lot of error messages, such as "no such pid." This is a normal part of the process. The shutdown code is using the pid files from daemons that were running when the backup was made to shut down daemons that were not started on the last boot. Of course there's no such pid.

Your system should come up normally, with a lot fewer errors than it had before; ideally no errors. The acid test of how well your restore works is to verify all packages. During the first stage backup, a verification was performed on the system, producing the file `rpmVa.txt`. Verify your system again, and compare the results to the one made earlier. E.g.:

```
bash# rpm -Va | sort +2 -t ' ' | uniq > ~/foo.txt
diff /mnt/zip/metadata/rpmVa.txt ~/foo.txt
```

Prelinking error messages are normal and you can ignore them. Do not first run the command **/etc/cron.daily/prelink** to remove them. Doing so may introduce new errors in the verification results that will skew your results.

Some files, such as configuration and log files, will have changed in the normal course of things, and you should be able to mentally filter those out of the report. Emacs users should check out its ediff facilities.

Now you should be up and running. It is time to test your applications, especially those that run as daemons. The more sophisticated the application, the more testing you may need to do. If you have remote users, disable them from using the system, or make it "read only" while you test it. This is especially important for databases, to prevent making any corruption or data loss worse than it already might be.

If you normally boot to X, it was disabled as part of the first stage restoration. Test X before you re-enable it. Re-enable it by changing that one line in `/etc/inittab`. Find the line that looks like this:

```
id:3:initdefault:
```

and change it to this:

```
id:5:initdefault:
```

Or just run this on the target to change it back. Note: different distributions use different run level schemes. These values work on Red Hat derived distributions; they may not on others.

```
sed -i s/id:.:initdefault:/id:5:initdefault:/g /etc/inittab
```

You should now be ready for rock and roll -- and some aspirin and a couch.

7. Distribution Specific Notes

Below are distribution notes from past experiences. If you have additional notes that you would like to add for other distributions, please forward them to me.

7.1. Debian/Ubuntu

The largest change for Debian based systems is the lack of anything analogous to RPM's integrity checking capability, **rpm -Va**, which we use to check for correct restoration. **debsums** is close but has other problems. You must install **debsums**, as soon after you have installed the system as possible. You must then run **apt-get clean && debsums_init**. This command will force the reinstallation of packages for which checksums have not already been generated. Package reinstallation can affect configuration files. Also see the caveats section of the **debsums** man page.

The most practical problem is that Debian and Ubuntu use UUIDs to identify partitions, rather than device files or labels (like Red Hat systems). We can preserve the UUIDs, and assign them to ext2/3/4 and swap partitions. Actually, we can get UUIDs for any partition **blkid** supports.

When using a Debian live CD, create a root password with **sudo passwd root**. Then **su -** as usual. Packages you may want to install: **apt install cryptsetup lvm2 grub2-common**. If the screen saver locks you out, log in again as user/live.

7.2. Fedora

The scripts now reflect Fedora 7, so you should not have to make any changes to these scripts.

> **Note:** I tested the above on a fresh installation of FC3. I had problems with devices after booting when I worked with a system that had been upgraded from FC2 to FC3.

7.3. Knoppix

I used to use Knoppix (http://www.knoppix.org/). Pasi Oja-Nisula (mailto:pon at iki dot fi) reports:

> For me the best thing about using Knoppix is that I don't need a specific boot medium for each machine, but I can use the same tools all the time. And hardware support in Knoppix is really great. I don't have that much experience with different platforms, but all the machines I've tried have worked fine, scsi drivers are found and so on.
>
> I'm doing this recovery thing by copying the backups over the network to other machine. The restore involves booting the Knoppix cd, fetching the metadata.tar.gz from the network machine. Then make.dev, mount.dev, fetching the other tar.gz files, grub and reboot. Some typing involved but thanks to your scripts it's quite straighforward. Unless changing from ide to scsi or something, but even then it's not that difficult, since Linux is easy to restore to different hardware.

Knoppix (http://www.knoppix.org/) detects USB devices for you, which is really nice. They make excellent (and roomier) substitutes for the ZIP drive.

Also see “System recovery with Knoppix” (http://www.ibm.com/developerworks/linux/library/l-knopx/index.html).

Do your restore as user “root” rather than as user “knoppix”. Otherwise you may get some directories and files owned by an oddball user or group. Also, for Knoppix (http://www.knoppix.org/), we tar the first stage stuff saving numeric user & group values instead of by name. The names may point to different numbers on Knoppix, so we would be restoring the files with incorrect user and group IDs.

7.4. Finnix

Finnix (http://www.finnix.org/) has some of the same advantages of Knoppix. In addition, it runs in command line mode with mouse support, which is great for the task at hand. It’s small, under 600 MB as of this writing, so you may be able to remaster it with your first stage data on it. It boots quickly. And it has LVM support. And Zile, a subset of Emacs. It is small enough that you can load it into RAM (**finnix toram** at boot) and use the CD-ROM drive. I am pleased with Finnix (http://www.finnix.org/) for this use, and it is now my standard first stage restoration Linux.

7.5. Grml

Grml (https://grml.org/) is another small Linux distribution aimed at system repair and recovery. It works well with these scripts. One potential gotcha: Grml defaults to zsh, and these scripts are written for bash. If zsh is a problem, Grml includes bash, so simply call bash and go from there.

Grml has an advantage for Debian users: it is based on Debian. More important, Debian includes a nice package, **grml-rescueboot**. This will put a Grml ISO in `/boot/grml` and add it to the GRUB boot menu, producing a nice rescue environment. Add the "toram" linux command line option will move Grml into RAM, making it possible to wipe the drive completely.

8. Application Specific Notes

Here are some notes about backing up particular applications.

8.1. Logical Volume Manager

Handling logical volumes turns out to be a bit of a trick: use the Finnix (http://www.finnix.org/) distribution’s startup code to turn LVM on and off. This results in distribution specific code for the first stage of restoration. It is generated in `make.fdisk`. To edit it, search `make.fdisk` for “Hideous”.

LVM required the addition of two new LVM specific scripts, **make.lvs** and **mount.lvs**. They are only generated and used if there are logical volumes present.

8.2. LUKS

LUKS (https://gitlab.com/cryptsetup/cryptsetup) allows a user to encrypt a partition. One can then use LVM to install multiple partitions in the encrypted partition.

Warning

When rebuilding a LUKS partition, you must know the passphrase! We do not handle that!

8.3. systemd

systemd (https://fedoraproject.org/wiki/Systemd) is a replacement for the System V init scripts formerly common on Linux. The main impact is that systemd replaces System V run levels with targets. See Boot Kernel Command Line (https://fedoraproject.org/wiki/Systemd#Boot_Kernel_Command_Line) for how to edit the command line when you boot after first stage restoration.

8.4. Selinux

Selinux is disabled on the test machines. `/selinux` is not backed up in any of these scripts. At a guess, you should probably disable selinux after the first stage restoration, and you will probably have some selinux specific tasks to perform before turning it back on.

8.5. Tripwire

If you run Tripwire or any other application that maintains a database of file metadata, rebuild that database immediately after restoring.

8.6. Squid

Squid is a HTTP proxy and cache. As such it keeps a lot of temporary data on the hard drive. There is no point in backing that up. Insert "--exclude /var/spool/squid" into the appropriate tar command in your second stage backup script. Then, get squid to rebuild its directory structure for you. Tack onto the tail end of the second stage restore script a command for squid to initialize itself. Here is how I did it over SSH in `restore`:

```
ssh $target "mkdir /var/spool/squid ; chown squid:squid /var/spool/squid;\
      /usr/sbin/squid -z;touch /var/spool/squid/.OPB_NOBACKUP"
```

The last command creates a file of length 0 called .OPB_NOBACKUP. This is for the benefit of Arkeia, and tells Arkeia not to back up below this directory

8.7. Arkeia

These notes are based on testing with Arkeia 4.2.

Arkeia (http://www.arkeia.com/) is a backup and restore program that runs on a wide variety of platforms. You can use Arkeia as part of a bare metal restoration scheme, but there are two caveats.

The first is probably the most problematic, as absent any more elegant solution you have to hand select the directories to restore in the navigator at restoration time. The reason is that, apparently, Arkeia has no mechanism for not restoring

files already present on the disk, nothing analogous to **tar**'s -p option. If you simply allow a full restore, the restore will crash as Arkeia over-writes a library which is in use at restore time, e.g. `lib/libc-2.1.1.so`. Hand selection of directories to restore is at best dicey, so I recommend against it.

The second caveat is that you have to back up the Arkeia data dictionary and/or programs. To do that, modify the `save.metatdata` script by adding Arkeia to the list of directories to save:

```
# arkeia specific:
tar cf - usr/knox | gzip -c > $zip/arkeia.tar.gz
```

You *must* back up the data dictionary this way because Arkeia does not back up the data dictionary. This is one of my complaints about Arkeia, and I have solved it in the past by saving the data dictionary to tape with The TOLIS Group's BRU (http://www.estinc.com/).

The data dictionary will be restored in the script `restore.metadata` automatically.

8.8. Amanda

Amanda (http://www.amanda.org/) (The Advanced Maryland Automatic Network Disk Archiver) works quite well with this set of scripts. Use the normal Amanda back-up process, and build your first stage data as usual. Amanda stores the data on tape in GNU tar or cpio format, and you can recover from individual files to entire backup images. The nice thing about recovering entire images is that you can then use variants on the scripts in this HOWTO to restore from the images, or direct from tape. I was able to restore my test machine with the directions from W. Curtis Preston's *Unix Backup & Recovery* (http://www.oreilly.com/catalog/unixbr/). For more information on it, see the Resources. The Amanda chapter from the book is on line (http://www.backupcentral.com/components/com_mambowiki/index.php/AMANDA).

I made two changes to the script `restore`. First, I changed it to accept a file name as an argument. Then, since Amanda's **amrestore** decompresses the data as it restores it, I rewrote it to cat the file into the pipe instead of decompressing it.

The resulting line looks like this:

```
cat $file | ssh $target "umask 000 ; cd / ; tar -xpkf - "
```

where **$file** is the script's argument, the image recovered from the tape by **amrestore**.

Since the command line arguments to **tar** prohibit over-writing, restore from images in the *reverse* of the order in which they were made. Restore most recent first.

Amanda may require setting ownership by hand if you back up the amanda data directory with `save.metadata`. Something like:

```
bash# chown -R amanda:disk /var/lib/amanda
```

You can also add that line to your scripts for second state restoration, such as `restore`.

`save.metadata` now has support for backing up enough of Amanda to use **amrecover**. However, this may mean you have to run `save.metadata` often in order to keep the saved data in synch with amanda's backups.

8.9. NTFS

OK, NTFS isn't an application. It is a file system used by Microsoft operating system Windows NT™ and its descendents. You can back it up and restore to it from Linux with **ntfsclone**, one of the NTFS utilities in the ntfsprogs suite, available from http://www.linux-ntfs.org/.

These scripts will create NTFS partitions, but will not put a file system on them. It is not clear from the docs whether **ntfsclone** will lay down a file system on a virgin partition or not.

8.10. UEFI

UEFI systems have their own hazards. If you dual boot with Windows™, you should probably rebuild Windows™; first, then rebuild Linux. This will require deleting making and possibly mounting NTFS partitions from the `make.dev.*` scripts.

Restoring the EFI partition is problematic. If the machine has Linux only, the current code should do it. If the machine dual boots, restoring the EFI partition with this code may not work. You may prefer to use other tools to restore the EFI partition.

9. Some Advice for Disaster Recovery

You should take your ZIP disk for each computer and the printouts you made, and place them in a secure location in your shop. You should store copies of these in your off-site backup storage location. The major purpose of off-site backup storage is to enable disaster recovery, and restoring each host onto replacement hardware is a part of disaster recovery.

You should also have several restoration Linux floppies or CD-ROMS, and possibly some ZIP drives in your off-site storage as well. Also, have copies of the rescue linux distribution on several of your computers so that they back each other up.

You should probably have copies of this HOWTO, with your site-specific annotations on it, with your backups and in your off-site backup storage.

10. What Now?

This HOWTO results from experiments on one computer. No doubt you will find some directories or files you need to back up in your first stage backup. I have not dealt with saving and restoring X on the first stage, nor have I touched at all on processors other than AMD or Intel.

I would appreciate your feedback as you test and improve these scripts on your own computers. I also encourage vendors of backup software to document how to do a minimal backup of their products. I'd like to see the whole Linux community sleep just a little better at night.

10.1. To Do

Volunteers are most welcome. Check with me before you start on one of these in case someone else is working on it already.

- Use `partprobe` after restoring the partition table, and after restoring the partitions?
- A partition editor to adjust partition boundaries in the `dev.hdx` file. This will let users adjust partitions for a different hard drive, or the same one with different geometry, or to adjust partition sizes within the same hard drive. A GUI would probably be a good idea here. On the other tentacle, the FSF's `parted` (http://www.gnu.org/software/parted) looks like it will fill part of the bill. It does re-size existing partitions, but with restrictions.
- `make.fdisk` currently only recognizes some FAT partitions, not all. Add code to `make.fdisk` or `make.gdisk` to recognize others and make appropriate instructions to rebuild them in the output files.
- For FAT12 or FAT16 partitions we do not format or write zeros into the partition so that MS-DOS 6.x does not get confused. See the notes on **fdisk** for an explanation of the problem. We do, however, preserve and restore UEFI partitions on drives with GUID partition tables (GPT). These are identified by partition code EF00. This requires the external utility `fsarchiver`.
- Translations into other (human) languages.
- I've referred to Red Hat Package Manager (rpm) from time to time. What are the equivalent deb commands?
- Modify the first stage backup code to only save the current kernel.

11. The Scripts

See the notes in the beginning of each script for a summary of what it does.

11.1. First Stage

11.1.1. `make.fdisk`

This script, run at backup time, creates scripts similar to `make.dev.hda` and `mount.dev.x`, below, for you to run at restore time. It also produces data files similar to `dev.hda` and `dev.hda.sfd`, below. The names of the scripts and data files produced depend on the device given this script as a parameter. Those scripts, run at restore time, build and mount the partitions on the hard drive. `make.fdisk` is called from `save.metadata`, below.

> **Warning**
>
> `make.fdisk` uses the older **fdisk** program, suited to MS-DOS partition tables. When possible and appropriate, use **make.gdisk**, below, instead.

```
#! /usr/bin/perl

# A perl script to create a script and input file for fdisk to
# re-create the partitions on the hard disk, and format the Linux and
# Linux swap partitions. The first parameter is the fully qualified
```

```
# path of the device of the hard disk, e.g. /dev/hda. The two
# resulting files are the script make.dev.x and the data file dev.x
# (where x is the hard drive described, e.g. hda, sdc). make.dev.x is
# run at restore time to rebuild hard drive x, prior to running
# restore.metadata. dev.x is the input file for fdisk.

# The directory tree where everything is put must already exist and be
# specified in the environment variable $zip.

# Copyright 2001 through the last date of modification Charles Curley
# except for the subroutine cut2fmt.

# cut2fmt Copyright (c) 1998 Tom Christiansen, Nathan Torkington and
# O'Reilly & Associates, Inc.  Permission is granted to use this code
# freely EXCEPT for book publication.  You may use this code for book
# publication only with the explicit permission of O'Reilly &
# Associates, Inc.

# This program is free software; you can redistribute it and/or modify it
# under the terms of the GNU General Public License as published by the
# Free Software Foundation; either version 2 of the License, or (at your
# option) any later version.

# This program is distributed in the hope that it will be useful, but
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
# General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
# 02110-1301, USA

# In addition, as a special exception, Tom Christiansen, Nathan
# Torkington and O'Reilly & Associates, Inc.  give permission to use
# the code of this program with the subroutine cut2fmt (or with
# modified versions of the subroutine cut2fmt that use the same
# license as the subroutine cut2fmt), and distribute linked
# combinations including the two.  You must obey the GNU General
# Public License in all respects for all of the code used other than
# the subroutine cut2fmt.  If you modify this file, you may extend
# this exception to your version of the file, but you are not
# obligated to do so.  If you do not wish to do so, delete this
# exception statement and the subroutine cut2fmt from your version.

# You can also contact the Free Software Foundation at
# http://www.fsf.org/

# Changes:

# 2017-07-29: Lots of small changes after testing on Debian 9.

# 2017-07-17: A new format string for more modern versions of
```

```
# fdisk. Using split instead of unpack for the output of fdisk. The
# first go at preserving LUKS partitions. Lots of other changes due to
# work on debian 9, stretch.

# 2017-07-16: Some disties (Debian 8,9; jessie,stretch), have
# lvm2. Test for it & use it if it's there.

# 2011-09-07: Per email from Michael Coffman: getextopts was leaving
# the "needs_recovery" flag in the ext? options, and mke2fs throws up
# on it. Added a line to remove it.

# 2010-03-22: We now use dumpe2fs to recover ext[3|4] partitions'
# features and use the -O option to mke2fs. So custom partitions' file
# systems will be laid down correctly.

# 2010-03-20: Improved handling of ext4. (There's more to do.) Fixed a
# bug with line endings in the output of "mount -l".

# 2010-01-12: An even better way to get UUIDs, which simplified the
# code and removed a bug, a false return if there was no UUID.

# 2009-12-24: Found a better way to get UUIDs. So we now provide them
# for swap partitions.

# 2007-11-05: Preserve UUIDs (Ubuntu). There is no way to set a UUID
# on a swap partition, so we punt: fix them up manually after the
# fact. The easiest fix is for you to edit fstab to refer to swap
# partitions by device name. Alternatively, see
# https://bugs.launchpad.net/ubuntu/+source/util-linux/+bug/66637,
# http://www.thinkwiki.org/wiki/Installing_Ubuntu_7.04_on_a_ThinkPad_T43#Swap_and_Hibernation_proble
# and
# https://bugs.launchpad.net/ubuntu/+source/util-linux/+bug/66637/comments/23
# et seq. for a list of things to do. Gnrrr.

# 2007-06-10: In addition to scanning /etc/fstab for LVM partitions
# (logical volumes), we also check the device files in /dev. This is
# because some logical volumes may be mounted by label, and scanning
# fstab won't pick those up.

# 2007-05-22: Changes for FHS compliance. Removed commented out
# references to ZIP drives. N.B.: we now take the location of where to
# put things as an environment variable, $zip.

# 2006-04-15: Added support for partition type 0x12, "Compaq
# diagnostic". This type is used for so-called "hidden diagnostics"
# partitions, e.g. on Lenovo/IBM computers.

# 2006-04-08: Primitive LVM support. It is kludgy in that it uses
# first stage restoration distribution (finnix) specific code to turn
# LVM on and off, but otherwise seems to work.

# 2006-03-28: We have a problem if swap partitions have
# labels. There's no way to retrieve the label from a swap
```

```
# partition. If we have one & only one swap partition, then we can
# pull it out of /etc/fstab. Otherwise the user is on her own. We scan
# fstab for swap mount points that have labels for their devices. If
# there is one and only one, we assume that's it, otherwise pass.

# 2005-10-29: We now provide the geometry as an argument to fdisk
# (which does not work on tomsrtbt). We also save data for sfdisk, and
# write out make.dev.xxx so that it will use sfdisk if it finds it.

# 2005-08-14: Due to experience on Knoppix, we now add the code to
# change the partition types to the end of the fdisk input file
# instead of right after creating the partition.

# 2004 04 10: fdisk v > 2.11 has wider columns. Added code to select
# the appropriate cut string based on fdisk's version.

# 2004 04 09: Added support for Mandrake's idea of devfs. On Mandrake,
# everything is mounted with devfs. So the mount devices are buried
# deep in places like /dev/ide/host0/bus0/target0/lun0/part1 instead
# of places like /dev/hda1, where $DEITY intended they should be. We
# have to reverse from the long devfs device to the shorter old style
# that tomsrtbt uses. The alternative is to keep track in an array of
# which devfs device belongs to which short device.

# 2003 12 29: Changed the regex for detecting whether a file system is
# read-write in the code that builds the mount file(s). The old test
# does not work if mount returns multiple parameters in the 5th field,
# e.g. (rw,errors=remount-ro) on some debian systems. This regex
# assumes that the rw parameter is always listed first, which may not
# always be the case. If it fails, take out the '\('. Thanks to Pasi
# Oja-Nisula <pon at iki dot fi> for pointing this out.

# 2003 01 09: Added support for FAT32. We now create two scripts for
# each hard drive, make.dev.[as]dx and mount.dev.[as]dx. These create
# and make file systems on each partition, and make mount points and
# mount them.

# 2002 12 25: added support to handle W95 extended (LBA) (f) and W95
# FAT 32 partitions. I have tested this for primary but not logical
# partitions.

# 2002 09 08: Added minimal support for ext3fs. We now detect mounted
# ext3fs partitions & rebuild but with no options. The detection
# depends on the command line "dumpe2fs <device> 2>/dev/null | grep -i
# journal" producing no output for an ext2fs, and output (we don't
# care what) for an ext3fs.

# This could stand extension to support non-default ext3 options such
# as the type of journaling. Volunteers?

# 2002 07 25: Bad block checking is now a command line option (-c) at
# the time the product script is run.
```

```
# 2002 07 03: Corrected the mechanism for specifying the default
# drive.

# 2001 11 25: Changed the way mke2fs gets its bad block
# list. badblocks does not guess at the block size, so you have to get
# it (from dumpe2fs) and feed it to badblocks. It is simpler to just
# have mke2fs call badblocks, but you do loose the ability to have a
# writing test easily. -- C^2

# 2001 11 25: Changed the regex that extracts partition labels from
# the mount command. This change does not affect the results at all,
# it just makes it possible to use Emacs' perl mode to indent
# correctly. I just escaped the left bracket in the regex. -- C^2

# Discussion:

# fdisk will spit out a file of the form below if you run it as "fdisk
# -l".

# root@tester ~/bin $ fdisk -l /dev/hda

# Disk /dev/hda: 64 heads, 63 sectors, 1023 cylinders
# Units = cylinders of 4032 * 512 bytes

#    Device Boot    Start       End    Blocks   Id  System
# /dev/hda1             1         9     18112+  83  Linux
# /dev/hda2            10      1023   2044224    5  Extended
# /dev/hda5            10       368    723712+  83  Linux
# /dev/hda6           369       727    723712+  83  Linux
# /dev/hda7           728       858    264064+  83  Linux
# /dev/hda8           859       989    264064+  83  Linux
# /dev/hda9           990      1022     66496+  82  Linux swap

# Or, on Debian 13 (trixie), using a more modern fdisk,

# root@testing:~# fdisk -l /dev/vda
# Disk /dev/vda: 6 GiB, 6442450944 bytes, 12582912 sectors
# Units: sectors of 1 * 512 = 512 bytes
# Sector size (logical/physical): 512 bytes / 512 bytes
# I/O size (minimum/optimal): 512 bytes / 512 bytes
# Disklabel type: gpt
# Disk identifier: 8D2B2C7A-E38F-4211-B96C-07BFFF0C8C9C

# Device       Start      End Sectors  Size Type
# /dev/vda1     2048  1157119 1155072  564M EFI System
# /dev/vda2  1157120  2760703 1603584  783M Linux filesystem
# /dev/vda3  2760704 12580863 9820160  4.7G Linux filesystem
# root@testing:~#

# What fdisk does not do is provide output suitable for later
# importing into fdisk, a la sfdisk. This script parses the output
# from fdisk and creates an input file for fdisk. Use the input file
# like so:
```

```
# fdisk /dev/hdx < dev.hdx

# For the bare metal restore package, this script also builds a script
# that will execute the above command so you can run it from your zip
# disk. Because the bare metal restore scripts all are in /root/bin,
# the data file and script created by this script are also placed
# there. The same script also creates appropriate Linux file systems,
# either ext2fs, or Linux swap. There is limited support for FAT12,
# FAT16 and FAT32. For anything else, you're on your own.

# Note for FAT32: According to the MS KB, there are more than one
# reserved sectors for FAT32, usually 32, but it can vary. Do a search
# in M$'s KB for "boot sector" or BPB for the gory details. For more
# info than you really need on how boot sectors are used, see
# http://support.microsoft.com/support/kb/articles/Q140/4/18.asp

# You can also edit dev.x to change the sizes of partitions. Don't
# forget, if you change the size of a FAT partition across the 32MB
# boundary, you need to change the type as well! Run "fdisk /dev/hda"
# or some such, then the l command to see the available partition
# types. Then go ahead and edit dev.x appropriately. Also, when moving
# partition boundarys with hand edits, make sure you move both logical
# and extended partition boundaries appropriately.

# Bad block checking right now is a quick read of the partition. A
# writing check is also possible but more difficult. You have to run
# badblocks as a separate command, and pass the bad block list to
# mke2fs in a file (in /tmp, which is a ram disk). You also have to
# know how large the blocks are, which you learn by running
# dumpe2fs. It gets messy and I haven't done it yet. You probably
# don't need it for a new hard drive, but if you have had a hard drive
# crash on you and you are reusing it (while you are waiting for its
# replacement to come in, I presume), then I highly recommend it. Let
# me know how you do it.

# For more information contact the author, Charles Curley, at
# http://www.charlescurley.com/.

use UUID;                       # for validating uuids.

# cut2fmt figures out the format string for the unpack function we use
# to slice and dice the output from fdisk. From Christiansen and
# Torkington, Perl Cookbook 5.

sub cut2fmt {
    my (@positions) = @_;
    my $template    = '';
    my $lastpos     = 1;

    foreach $place (@positions) {
        $template .= "A" . ($place - $lastpos) . " ";
        $lastpos = $place;
```

```
    }

    $template .= "A*";
    return $template;
}


# Sub gpl, a subroutine to ship the GPL and other header information
# to the current output file.

sub gpl {
    my $FILE = shift;
    my $year = shift;

    print $FILE <<FINIS;

# Copyright $year through the last date of modification Charles Curley.

# This program is free software; you can redistribute it and/or modify it
# under the terms of the GNU General Public License as published by the
# Free Software Foundation; either version 2 of the License, or (at your
# option) any later version.

# This program is distributed in the hope that it will be useful, but
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
# General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
# 02110-1301, USA

# You can also contact the Free Software Foundation at http://www.fsf.org/

# For more information contact the author, Charles Curley, at
# http://www.charlescurley.com/.

FINIS

}

sub getBootSector {
    my $infile = $_[0];
    my $outfile = $_[1];

    $systemcmd = "dd if=$infile of=$outfile bs=512 count=1 &> /dev/null ";
    system ($systemcmd);
}


# If we have one & only one swap partition, then this must be
# it. Otherwise the user is on her own. We scan fstab for swap mount
```

```
# points that have labels for their devices. If there is one and only
# one, we assume that's it, otherwise pass.

sub getswaplabel {
    my $dev = $_[0];

    open (FSTAB, "< /etc/fstab")
        or die "Couldn't fork: $!\n";
    while (defined (my $line = <FSTAB>)) {
        chomp ($line);
        @fstabs = split (" ", $line);
        if (@fstabs[1] eq "swap") {
            $swaplabel = @fstabs[0];
            if ($swaplabel =~ /LABEL/) {
                $swaps++;
                $sl = substr ($swaplabel, 6);
            }
#           print ("\"@fstabs[0]\", \"@fstabs[1]\", \"$sl\", $swaps.\n");
            break;
        }
    }
    close (FSTAB);

    # print ("label is $sl.\n");

    if ($swaps == 1) {
        $ret = "-L $sl $dev\n";
    } else {
        $ret = "$dev\n";
    }

#   print ("Returning :$ret\n");

    return $ret;
}

# doluks, for handing LUKS partitions.

sub doluks {
    my $device = shift;
    $devName=substr ($device, 5);
    $cryptName = "${devName}_crypt";

    $fileName = "${outputfilepath}metadata/$devName-header-backup";
    if ( -e $fileName) {
        unlink ($fileName);
    }
    if (system ("cryptsetup luksHeaderBackup $device --header-backup-file $fileName") != 0 ) {
        die ("LUKS backup of $device failed!\n\n");
    }
    $format .= "echo Device $device is a LUKS partition.\n";
    $format .= "cryptsetup luksHeaderRestore $device --header-backup-file ../metadata/$devName-heade
    $format .= "cryptsetup isLuks $device\n";
```

```
    $format .= "if [ \"\$?\" != '0'  ] ; then\n";
    $format .= "    echo Error: $device is NOT a LUKS partition.\n";
    $format .= "    exit\n";
    $format .= "fi\n\n";
    $format .= "echo Opening $device as a LUKS partition.\n";
    $format .= "cryptsetup luksOpen $device $cryptName\n";
    $format .= "ls /dev/mapper\n";

    print ("\n\nWarning!! $device is a LUKS partition. Be sure you have the passphrase for restorati

    dolvm ();
}


# dolvm is a subroutine to handle LVM partitions.

$lvms = 0;                      # true if we've been here before

sub dolvm {

#     print ("In dolvm ()...\n");

    if ($lvms == 0) {
        $lvms = 1;

        # Scan /etc/fstab for the logical volumes and write a script to
        # make file systems on them and another to mount 'em later on.

        $mklvs = open (MKLVS, "> ${outputfilepath}bin/make.lvs")
            or die "Couldn't fork: $!\n";

        print MKLVS <<FINIS;
#! /bin/sh

# A script to create file systems on logical volumes. Created at bare
# metal backup time by the Perl script make.fdisk.
FINIS

        &gpl (*MKLVS, "2006");


        print MKLVS <<FINIS;

export blockcheck=\$1;

if [ "\$blockcheck" != "-c" ] && [ -n "\$blockcheck" ]
then
    echo "\${0}: Build file systems on logical volumes."
    echo "\${0}: -c: block check during file system making."
    exit 1;
fi

FINIS
```

```
        $mtlvs = open (MTLVS, "> ${outputfilepath}bin/mount.lvs")
            or die "Couldn't fork: $!\n";

        print MTLVS <<FINIS;
#! /bin/sh

# A script to mount file systems on logical volumes. Created at bare
# metal backup time by the Perl script make.fdisk.
FINIS

        &gpl (*MTLVS, "2007");


        # Now cycle through all the known logical volumes & set them
        # up. N.B.: This has been tested on machines with only one LV
        # and with two. It picks up both on that test machine.

        $pvdisp = open (PVDISP, "pvdisplay -c |")
            or die ("Can't open LVM display.\n");
        while (defined (my $pv = <PVDISP>)) {
            chomp ($pv);
#             print ("$pv\n");
            @pv = split (":", $pv);
            $uid = @pv[11];
            $pvname = @pv[1];
            $phv = @pv[0];
#             print ("pv $pvname has uid $uid.\n");

            # back up the LVM's lvm details. Get the config files.
            if (system ("vgcfgbackup -f ${outputfilepath}metadata/LVM.backs.$pvname $pvname > /dev/n
                die ("Failed to back up $pvname!!\n");
            }

            print (MKLVS "echo \"y\\n\" | pvcreate -ff --uuid \"$uid\"\\\n");
            print (MKLVS "    --restorefile ../metadata/lvm/backup/${pvname} $phv\n");
            print (MKLVS "vgcfgrestore --file ../metadata/LVM.backs.$pvname $pvname\n\n");
        }

        print (MKLVS "# Hideously disty dependent! turn on LVM.\n");
        print (MKLVS "if [ -e /etc/init.d/lvm ] ; then\n");
        print (MKLVS "    /etc/init.d/lvm start\nfi\n\n");
        print (MKLVS "if [ -e /etc/init.d/lvm2 ] ; then\n");
        print (MKLVS "    /etc/init.d/lvm2 start\nfi\n\n");

        # Now tell the volume manager software to find the new volume:

        print (MKLVS "# scan for new volume groups\n");
        print (MKLVS "vgscan --mknodes\n\n");
        print (MKLVS "# Activate the volume groups\n");
        print (MKLVS "vgchange -ay\n\n");
        print (MKLVS "ls /dev/mapper\n\n");
```

```
# Now walk fstab in search of logical volumes. This is
# necessary to pick up swap partitions, and it may pick up
# others. We need fstab below to match the partitions up with
# their mount points, so we keep the array around.

%volsfound = ();
open (FSTAB, "< /etc/fstab")
    or die "Couldn't fork: $!\n";
@fstab = <FSTAB>;
foreach $line (@fstab) {
    @fstabs = split (" ", $line);
    #                   Red Hat|Ubuntu/Debian
    if (@fstabs[0] =~ /VolGroup|mapper/ ) {
        # print ("$line");
        if (@fstabs[2] eq "swap") {
            print (MKLVS "echo\necho making LV @fstabs[0] a swap partition.\n");
            print (MKLVS "mkswap \$blockcheck @fstabs[0]\n\n");
        } elsif (@fstabs[2] == "ext4") {
            print (MKLVS "echo\necho making LV @fstabs[0], mounted on @fstabs[1],");
            print (MKLVS " an ext4 partition.\n");
            print (MKLVS "mke2fs -t ext4" . getextopts (@fstabs[0]));
            print (MKLVS "\$blockcheck @fstabs[0]\n\n");

            print (MTLVS "mkdir -p /target$fstabs[1]\n");
            print (MTLVS "mount @fstabs[0] /target$fstabs[1]\n\n");
            $volsfound{@fstabs[0]} = 4;
        } elsif (@fstabs[2] == "ext3") {
            print (MKLVS "echo\necho making LV @fstabs[0], mounted on @fstabs[1],");
            print (MKLVS " an ext3 partition.\n");
            print (MKLVS "mke2fs -t ext3" . getextopts (@fstabs[0]));
            print (MKLVS "\$blockcheck @fstabs[0]\n\n");

            print (MTLVS "mkdir -p /target$fstabs[1]\n");
            print (MTLVS "mount @fstabs[0] /target$fstabs[1]\n\n");
            $volsfound{@fstabs[0]} = 3;
        } elsif (@fstabs[2] == "ext2") {
            print (MKLVS "echo\necho making LV @fstabs[0], mounted on @fstabs[1],");
            print (MKLVS " an ext2 partition.\n");
            print (MKLVS "mke2fs \$blockcheck @fstabs[0]\n\n");

            print (MTLVS "mkdir -p /target$fstabs[1]\n");
            print (MTLVS "mount @fstabs[0] /target$fstabs[1]\n\n");
            $volsfound{@fstabs[0]} = 2;
        } else {
            print ("Opps, unknown type of logical volume, @fstabs[0]\n");
        }
    }
}

print (MKLVS "# End of fstab device list.\n\n");

#         print ("Volumes already found are: ");
#         while ( ($k, $v) = each %volsfound ) {
```

```
        #             print ("$k ==> $v ");
        #         }
        #         print ("\n");

        # Now walk the logical volume devices and pick up any partitions
        # formated ext3/ext2/ext4. This may result in duplicates if the
        # partitions have labels but are mounted by device name rather
        # than by label.

        # FIX ME: This loop doesn't distinguish between ext3 and
        # ext4. It may not need to, as we pull the options from the
        # partition itself. If so, that would be nice.

        opendir (DEVHANDLE, "/dev") or die ("Can't open /dev!!\n");
        while ( defined ($fname = readdir (DEVHANDLE))) {
            @fnames = (@fnames, $fname);
        }
        @sorted = sort (@fnames);
        foreach $fname (@sorted) {
            #                Red Hat|Ubuntu
            if ($fname =~ /^VolGroup|^mapper/ && -d "/dev/$fname") {
                # print ("Inside /dev is $fname.\n");
                opendir (VOLHANDLE, "/dev/$fname")
                    or die ("Can't open /dev/$fname!!\n");
                while ( defined ($vname = readdir (VOLHANDLE))) {
                    @vnames = (@vnames, $vname);
                }
                @vsorted = sort (@vnames);
                foreach $vname (@vsorted) {
                    # print ("/dev/$fname/$vname: " . $volsfound{"/dev/$fname/$vname"} . "\n");
                    if($vname ne "." && $vname ne ".."
                       && $volsfound{"/dev/$fname/$vname"} < 1) {
#                          print ("Inside /dev/$fname is $vname.\n");
                        my $journal = 0;

                        # FIX ME: add tests to be sure it's a symlink
                        # to a block device.

                        # Is it extX?
                        if (! system ("dumpe2fs /dev/$fname/$vname 1> /dev/null 2>&1")) {

                            # Is it extX?
                            open (DUMP, "dumpe2fs /dev/$fname/$vname 2> /dev/null|");

                            @lines = <DUMP>;

                            # if it is short, it is most likely an error message.
                            if (scalar (@lines) > 5) {

                                # If we've gotten here we have a valid
                                # ext[2|3|4] file system. Now prepare to
                                # spit out the commands to recreate
                                # it. Get the label, if any, and whether
```

```
                                # there is a journal or not.

                                foreach $_ (@lines) {
                                    if (/Filesystem volume name:/) {
                                        $label = substr ($_, 26);
                                        chop ($label);
#                                         print ("\$label is \"$label\".\n");
                                    }
                                    if (/has_journal/) {
                                        $journal = 1;
                                    }

                                }

                                # get the mount point from fstab so we can mount it.
                                $mountpoint = "none!";
                                foreach $fstab (@fstab) {
                                    @fstabs = split (" ", $line);
                                    if (@fstabs[0] eq "LABEL=$label" ) {
                                        $mountpoint = @fstabs[1];
                                        #                                       print ("mount point is
                                        last;
                                    }
                                }
                                if (length ($label) ) {
                                    $label = "-L \"" . $label . "\"";
                                }

                                if ($journal > 0) {
                                    print (MKLVS "echo\necho making LV /dev/$fname/$vname");
                                    print (MKLVS " an ext3/4 partition.\n");
                                    print (MKLVS "mke2fs" . getextopts ("/dev/$fname/$vname"));
                                    print (MKLVS "$label \$blockcheck /dev/$fname/$vname\n\n");

                                    if ($mountpoint ne "none!") {
                                        print (MTLVS "mkdir -p /target$mountpoint\n");
                                        print (MTLVS "mount /dev/$fname/$vname /target$mountpoint\n\
                                    }
                                } else {
                                    print (MKLVS "echo\necho making LV /dev/$fname/$vname");
                                    print (MKLVS " an ext2 partition.\n");
                                    print (MKLVS "mke2fs $label \$blockcheck /dev/$fname/$vname\n\n"

                                    if ($mountpoint ne "none!") {
                                        print (MTLVS "mkdir -p /target$mountpoint\n");
                                        print (MTLVS "mount /dev/$fname/$vname /target$mountpoint\n\
                                    }
                                }
                            }
                        }
                    }
                }
                closedir (VOLHANDLE);
```

```
            }
        }
        print (MKLVS "ls /dev/mapper\n");
        print (MTLVS "mount | grep -i \"/target\"\n");

        closedir (DEVHANDLE);

        close (FSTAB);
        close (MKLVS);
        close (MTLVS);

        chmod 0700, "${outputfilepath}bin/make.lvs";
        chmod 0700, "${outputfilepath}bin/mount.lvs";

        # Copy the LVM configuration to where we can get at it...
        system ("cp -rp /etc/lvm ${outputfilepath}metadata/");

    }

#     print ("Leaving dolvm ()...\n");

    return ($ret);
}

sub getuuid {
    my $device = shift;

    # print ("In getuuid. \$device = $device.\n");

    my $rs = open (UUID, "blkid -o value -s UUID $device |")
        or die ("Can't open volume id tool.\n");
    while (defined (my $uuid = <UUID>)) {
        chomp ($uuid);
        # print ("$uuid\n");

        if (UUID::parse($uuid, my $rawUuid) != -1) {
            # print ("Device $device has uuid '$uuid'.\n");
            return ($uuid);
        }
    }
    return ('');
}

# scan the partition for the options to mke2fs.

sub getextopts {
  my $device = $_[0];
  # print ("Device is $device\n");

  open (DUMP, "dumpe2fs $device 2> /dev/null|") or
    die ("Can't open a dump.\n");

  my @lines = <DUMP>;
```

```
  if (scalar (@lines) > 1) {

    foreach $_ (@lines) {
      # print ("line: $_");

      if (/Filesystem features/) {
        $_ = substr ($_, 26);
        chomp ();
        $_ =~ s/needs_recovery//g;
        $_ =~ s/metadata_csum//g;
        $_ = join (',', split (" "));
        # print ("File system features of $device are: |${_}|.\n");
        return ($_ = ' -O ' . $_ . ' ');
      }
    }
    die ("Couldn't get features from device $device\n");
  } else {
    die ("Couldn't run dumpe2fs on $device.\n");
  }
}

# Begin main line code.

$outputfilepath = $ENV{"zip"} . "/";
if ($outputfilepath eq "/") {
    $outputfilepath =  "./";
}
# print "\$outputfilepath is ${outputfilepath}\n";

# These will "fail" silently if the directory is already there, which
# it should be if we're called from save.metadata. These simplify
# testing, though.

mkdir ("${outputfilepath}metadata", 0700);
mkdir ("${outputfilepath}bin", 0700);

# Provide a default device.

# print "\$ARGV[0] is $ARGV[0].\n";

$device = defined ($ARGV[0]) ? $ARGV[0] : "/dev/hda";

# Need to check to see if $device is a sym link. If it is, the mount
# point is the target of the link. (Mandrake) Otherwise we search for
# mount points on $device. Fedora, Red Hat.

if ( -l $device) {

    # It is a sym link. Get the target of the link, then make it into
    # an absolute path, preserving the numbering.

    $mountdev = '/dev/' . readlink ($device);
    $mountdev =~ s|ide/host(\d+)/bus(\d+)/target(\d+)/lun(\d+)/disc
```

```
        |ide/host\1/bus\2/target\3/lun\4|x;
} else {
    # not a sym link; just assign it.
    $mountdev = $device;
}

# print "Device is $device; mount device is $mountdev.\n";

# Prepare format string. Here are two format strings I have found
# useful. Here, column numbers are 1 based, i.e. the leftmost column
# is column 1, not column 0 as in Emacs.

# We select a format string according to fdisk's version.

# Unfortunately, some time between version 2.12 and 2.25 (on Debian
# jessie, 8.x), the authors switched to a variable column width. So we
# use unpack for the earlier versions, and split for the later
# ones. We could probably get away with using split for them all, but
# I no longer have older versions with which to test. Volunteers?

# I would appreciate any help in nailing down the relevant version
# numbers.

$fdpid = open (FDVER, "fdisk -v |") or die "Couldn't fork: $!\n";
while (<FDVER>) {
    chomp;
    # print ("$_\n");
    # print "fdisk version is |$_|\n";
    $_ =~ s/[^\d.]//g; # strip non-numbers, non-periods, as in "2.12pre".
    $_ =~ m/(\d+\.\d+)/;
    $fdver = $1;
}

# print "fdisk version is |$fdver|\n";

$fmt="Unititialized!";
if ($fdver < 2.12) {
# fdisk to 2.11?? Red Hat, Fedora Core 1
    $fmt = cut2fmt (11, 19, 24, 34, 45, 49);
} elsif ($fdver < 2.20 ) {      # 2.25 on Debian jessie 8.x
    # fdisk 2.12 & up?? Mandrake 10.0, Fedora Core 2
    $fmt = cut2fmt (12, 14, 26, 38, 50, 55);
} else {
    # fdisk 2.41 on Debian 13, & up. 12??
    $fmt = cut2fmt (9, 17, 26, 34, 41, 75);
}
# print "Format string is $fmt.\n";

# define fields in the array @_.
$dev = 0;
$bootable = 1;
$firstcyl = 1;
$endcyl = 2;
```

```
$parttype = 5;
$partstring = 6;

$target = "\/target";

$outputfilename = $device;
$outputfilename =~ s/\//./g;
$outputfilename = substr ($outputfilename, 1, 100);

# Make a hash of the labels.
$mpid = open (MOUNT, "mount -l |") or die "Couldn't fork: $!\n";
while (<MOUNT>) {
    if ($_ =~ /^$mountdev/i) {  # is this a line with a partition in it?
        chomp;
        # print "mount line is \"$_\"\n"; # print it just for grins
        @_ = split;
        if ($_[6] ne "") {     # only process if there actually is a label
            $_[6] =~ s/[\[\]]//g;     # strike [ and ].
            $labels{$_[0]} = $_[6];
#           print "The label of file device $_[0] is $labels{$_[0]}.\n";
        }

        # foreach $el (@_) {
        #     print ("Element is $el\n");
        # }

        # We only mount if it's ext2fs, ext3fs, or ext4fs, and read
        # and write.

        if ($_[4] =~ /ext[234]/ and $_[5] =~ /\(rw/ ) {
            # print ("R/W ext[234] partition found.\n");
            if ($_[0] =~ /ide/i) {

                # We have a devfs system, e.g. Mandrake. This code
                # converts back from the devfs designation to the old
                # /dev/hd* designation for tomsrtb. I have NOT checked
                # this out for drives other than /dev/hda. Also, this
                # code does not handle SCSI drives.

                if ( $_[0] =~ /target0/ && $_[0] =~ /bus0/ ) {
                    $letter = 'a';
                } elsif ( $_[0] =~ /target1/ && $_[0] =~ /bus0/) {
                    $letter = 'b';
                } elsif ( $_[0] =~ /target0/ && $_[0] =~ /bus1/) {
                    $letter = 'c';
                } else {
                    $letter = 'd';
                }
                $_[0] =~ s|/ide/host\d+/bus\d+/target\d+/lun\d+/part|/hd|g;
                $_[0] =~ s/hd/hd$letter/;
            }
            $mountpoints{$_[2]} = $_[0];
            # print "$_[2] is the mountpoint for tomsrtbt|finnix";
```

```
            # print " device $mountpoints{$_[2]}.\n";
        }
    }
}
close (MOUNT);

# Get sfdisk output. If we have sfdisk at restore time (e.g. Knoppix),
# we'll use it.

system "sfdisk --dump $device > ${outputfilepath}metadata/${outputfilename}.sfd";

# Otherwise we'll use the output from fdisk, which may or may not be
# any more accurate.

$fpid = open (FDISK, "fdisk -l $device |") or die "Couldn't fork: $!\n";

open (OUTPUT, "> ${outputfilepath}metadata/${outputfilename}")
    or die "Couldn't open output file ${outputfilepath}metadata/${outputfilename}.\n";

while (<FDISK>) {
    if ($_ =~ /^$device/i) { # is this the line with our partition in it?
        $bootFlag = 0;

        # print $_;               # print it just for grins
        chomp;                      # kill trailing \r

        if ($fdver > 2.24) {
            # Unfortunately, split does not get rid of the bootable
            # flag asterisk for us.
            @_ = split;

            # Now we get rid of the bootable flag (and note it in passing).
            if (@_[$bootable] eq '*') {
                $bootFlag = $device;
                splice (@_, 1, 1);
            }
            # print ("using split on the output of fdisk.\n");
        } else {
            @_ = unpack ($fmt, $_);
            if (@_[$bootable] =~ /\*/) {
                $bootFlag = $device;
            }
        }

        # Now strip white spaces from cylinder numbers, white space &
        # leading plus signs from partition type.
        @_[$firstcyl] =~ s/[ \t]+//;
        @_[$endcyl] =~ s/[ \t]+//;
        @_[$partstring] =~ s/[ \t]+//;
        @_[$parttype] =~ s/[+ \t]+//;

        $partnumber = substr(@_[$dev], 8, 10); # get partition number for this line
```

```
# just for grins
# print "Partition: |$partnumber|, |@_[$firstcyl]|, |@_[$endcyl]|,";
# print " Type: |@_[$parttype]|, String: |@_[$partstring]|\n";

# foreach $el (@_) {
#     print ("Element is $el\n");
# }

# Here we start creating the input to recreate the partition
# this line represents.

print OUTPUT "n\n";
if ($partnumber < 5) {
    # primary Linux partition
    if (@_[$parttype] == 83) {
        print OUTPUT "p\n$partnumber\n@_[$firstcyl]\n";
        # in case it's all on one cylinder
        if (@_[$firstcyl] ne @_[$endcyl]) {
            print OUTPUT "@_[$endcyl]\n";
        }

        if (system ("cryptsetup isLuks @_[$dev]") == 0) {
            # print ("@_[$dev] is a logical LUKS partition! Woo Hoo!!\n");
            &doluks (@_[$dev]);
        } else {

            if ($labels{@_[$dev]}) { # do we have a label?
                $format .= "echo\necho formatting $checking@_[$dev]\n";
                # $format .= "mke2fs $ext3{$_[$dev]}\$blockcheck";
                # $format .= " -L $labels{@_[$dev]} @_[$dev]\n";
                $format .= "mke2fs" . getextopts ($_[$dev]) . "\$blockcheck";
                $format .= " -L $labels{@_[$dev]} @_[$dev]\n";
            } else {
                $format .= "echo\necho formatting $checking@_[$dev]\n";
                # $format .= "mke2fs $ext3{$_[$dev]}\$blockcheck @_[$dev]\n";
                $format .= "mke2fs" . getextopts ($_[$dev]) . "\$blockcheck @_[$dev]\n";
            }
            $uuid = getuuid ($_[$dev]);
            if (length ($uuid) > 0) {
                $format .= "tune2fs -U $uuid @_[$dev]\n";
            }

        }
        $format .= "\n";

        # extended partition
    } elsif (@_[$parttype] == 5) {
        # print ("Creating Extended Partition.\n");
        print OUTPUT "e\n$partnumber\n@_[$firstcyl]\n";
        if (@_[$firstcyl] ne @_[$endcyl]) {
            print OUTPUT "@_[$endcyl]\n";
        }
```

```
    # extended partition, Win95 Ext'd (LBA)
} elsif (@_[$parttype] eq "f") {
    # print ("Creating Extended LBA Partition.\n");
    print OUTPUT "e\n$partnumber\n@_[$firstcyl]\n";
    if (@_[$firstcyl] ne @_[$endcyl]) {
        print OUTPUT "@_[$endcyl]\n";
    }
    $typechanges .= "t\n$partnumber\nf\n";

    # primary Linux swap partition
} elsif (@_[$parttype] == 82) {
    print OUTPUT "p\n$partnumber\n@_[$firstcyl]\n";
    if (@_[$firstcyl] ne @_[$endcyl]) {
        print OUTPUT "@_[$endcyl]\n";
    }
    $typechanges .= "t\n$partnumber\n82\n";
    $format .= "echo\necho Making @_[$dev] a swap partition.\n";
    $format .= "mkswap \$blockcheck ";
    $uuid = getuuid ($_[$dev]);
    if (length ($uuid) > 0) {
        $format .= "-U $uuid ";
    }
    if ($labels{@_[$dev]}) { # do we have a label?
        $format .= "-L $labels{@_[$dev]} ";
        $format .= "@_[$dev]\n";
    } else {
        $format .= getswaplabel (@_[$dev]);
    }

    $format .= "\n";

    # Primary mess-dos partition. We don't handle hidden
    # partitions.

} elsif ( @_[$parttype] == 1 || @_[$parttype] == 4 || @_[$parttype] == 6
          || @_[$parttype] eq "b" || @_[$parttype] eq "c"
          || @_[$parttype] eq "e" || @_[$parttype] eq "12" ) {
    # print ("Making DOS primary partition.\n");

    getBootSector (@_[$dev], "${outputfilepath}metadata/$outputfilename$partnumber");

    print OUTPUT "p\n$partnumber\n@_[$firstcyl]\n";
    # in case it's all on one cylinder
    if (@_[$firstcyl] ne @_[$endcyl]) {
        print OUTPUT "@_[$endcyl]\n";
    }

    $typechanges .= "t\n$partnumber\n@_[$parttype]\n";
    $format .= "echo\necho formatting $checking@_[$dev]\n";
    $format .= "mkdosfs \$blockcheck";
    if ( @_[$parttype] == b || @_[$parttype] == c
         || @_[$parttype] eq "12" ) {
        # We have a W9x FAT32 partition. Add a command line switch.
```

```
            $format .= " -F 32";
        }
        $format .= " @_[$dev]\n";
        $format .= "# restore FAT boot sector.\n";
        $format .= "dd if=$outputfilename$partnumber";
        $format .= " of=@_[$dev] bs=512 count=1\n\n";

    } elsif ( @_[$parttype] == "8e") {
        $format .= dolvm ();
    } else {
        # anything else partition
        print OUTPUT "p\n@_[$firstcyl]\n";
        if (@_[$firstcyl] ne @_[$endcyl]) {
            print OUTPUT "@_[$endcyl]\n";
        }
        $typechanges .= "t\n$partnumber\n@_[$parttype]\n";
    }

} else {
    # logical Linux partition
    if (@_[$parttype] == 83) {
        print OUTPUT "l\n@_[$firstcyl]\n";
        if (@_[$firstcyl] ne @_[$endcyl]) {
            print OUTPUT "@_[$endcyl]\n";
        }

        if (system ("cryptsetup isLuks @_[$dev]") == 0) {
            # print ("@_[$dev] is a logical LUKS partition! Woo Hoo!!\n");
            doluks (@_[$dev]);
        } else {

            if ($labels{@_[$dev]}) { # do we have a label?
                $format .= "echo\necho formatting $checking@_[$dev]\n";
                # $format .= "mke2fs $ext3{@_[$dev]}\$blockcheck";
                # $format .= " -L $labels{@_[$dev]} @_[$dev]\n";
                $format .= "mke2fs" . getextopts ($_[$dev]) . "\$blockcheck";
                $format .= " -L $labels{@_[$dev]} @_[$dev]\n";
            } else {
                $format .= "echo\necho formatting $checking@_[$dev]\n";
                # $format .= "mke2fs $ext3{@_[$dev]}\$blockcheck @_[$dev]\n";
                $format .= "mke2fs" . getextopts ($_[$dev]) . "\$blockcheck @_[$dev]\n";
            }
            $uuid = getuuid ($_[$dev]);
            if (length ($uuid) > 0) {
                $format .= "tune2fs -U $uuid @_[$dev]\n";
            }
        }
        $format .= "\n";

        # logical Linux swap partition
    } elsif (@_[$parttype] == 82 ) {
        print OUTPUT "l\n@_[$firstcyl]\n";
        if (@_[$firstcyl] ne @_[$endcyl]) {
```

```
                print OUTPUT "@_[$endcyl]\n";
            }
            $typechanges .= "t\n$partnumber\n82\n";
            $format .= "echo\necho Making @_[$dev] a swap partition.\n";
            $format .= "mkswap \$blockcheck ";
            $uuid = getuuid ($_[$dev]);
            if (length ($uuid) > 0) {
                $format .= "-U $uuid ";
            }
            if ($labels{@_[$dev]}) { # do we have a label?
                $format .= "-L $labels{@_[$dev]} ";
                $format .= "@_[$dev]\n";
            } else {
                $format .= getswaplabel (@_[$dev]);
            }

            $format .= "\n";

            # Logical mess-dos partition. We don't handle hidden
            # partitions.

        } elsif ( @_[$parttype] == 1 || @_[$parttype] == 4 || @_[$parttype] == 6
                  || @_[$parttype] eq "b" || @_[$parttype] eq "c"
                  || @_[$parttype] eq "e" || @_[$parttype] eq "12" ) {
#           print ("Making DOS logical partition.\n");

            getBootSector (@_[$dev], "${outputfilepath}metadata/$outputfilename$partnumber");

            print OUTPUT "l\n$partnumber\n@_[$firstcyl]\n";
            # in case it's all on one cylinder
            if (@_[$firstcyl] ne @_[$endcyl]) {
                print OUTPUT "@_[$endcyl]\n";
            }
            $typechanges .= "t\n$partnumber\n@_[$parttype]\n";
            $format .= "echo\necho formatting $checking@_[$dev]\n";
            $format .= "mkdosfs \$blockcheck";
            if ( @_[$parttype] == b || @_[$parttype] == c
                 || @_[$parttype] eq "12" ) {
                # We have a W9x FAT32 partition. Add a command line switch.
                $format .= " -F 32";
            }
            $format .= " @_[$dev]\n";
            $format .= "# restore FAT boot sector.\n";
            $format .= "dd if=$outputfilename$partnumber";
            $format .= " of=@_[$dev] bs=512 count=1\n\n";

        } elsif ( @_[$parttype] == "8e") {
            $format .= dolvm ();
        } else {
            # anything else partition
            print OUTPUT "l\n@_[$firstcyl]\n";
            if (@_[$firstcyl] ne @_[$endcyl]) {
                print OUTPUT "@_[$endcyl]\n";
```

```
                }
                $typechanges .= "t\n$partnumber\n@_[$parttype]\n";
            }
        }

        # handle bootable partitions
        if ($bootFlag) {
            print OUTPUT "a\n$partnumber\n";
            print ("Partition $_[$dev] is bootable.\n");
        }
    } else {
        # If we got here, the current line does not have a partition in it.

        # Get the geometry for fdisk. Force fdisk to use the current
        # geometry at restoration time. Comment this out for
        # tomstrbt's fdisk; it doesn't like it.

        if ($_ =~ /heads.*sectors.*cylinders/i) {
#           print $_;               # again, for grins.
            chomp;
            @geometry = split (/ /, $_);
            $geometry = "-H $geometry[0] -S $geometry[2] -C $geometry[4]";
#           print $geometry;
        }
    }
}

# Append all the partition type changes, validate, and print out the
# results.

print OUTPUT "${typechanges}v\nw\n";

close (OUTPUT);
close (FDISK);


open (OUTPUT, "> ${outputfilepath}bin/make.$outputfilename")
    or die "Couldn't open output file ${outputfilepath}bin/make.$outputfilename.\n";

print OUTPUT <<FINIS;
#! /bin/sh

# A script to restore the partition data of a hard drive and format
# the partitions. Created at bare metal backup time by the Perl script
# make.fdisk.
FINIS

&gpl (*OUTPUT, "2001");

print OUTPUT <<FINIS;

swapoff -a
```

```
# Hideously disty dependent! Turn off LVM.
if [ -e /etc/init.d/lvm ] ; then
    /etc/init.d/lvm stop
fi
if [ -e /etc/init.d/lvm2 ] ; then
    /etc/init.d/lvm2 stop
fi

export blockcheck=\$1;

if [ "\$blockcheck" != "-c" ] && [ -n "\$blockcheck" ]
then
    echo "\${0}: automated restore with no human interaction."
    echo "\${0}: -c: block check during file system making."
    exit 1;
fi

FINIS

# Clean the old partition table out. Turn off swap in case we're using
# it.

# print OUTPUT "dd if=/dev/zero of=$device bs=512 count=2\n\nsync\n\n";
print OUTPUT "echo Zeroing out the hard drive. This may take a while.\n";
print OUTPUT "dd if=/dev/zero of=$device bs=10240 count=2000000\n\nsync\n\n";


# command for fdisk

$fdiskcmd .= "# see if we have sfdisk & if so use it.\n";
$fdiskcmd .= "if which sfdisk ; then\n";
$fdiskcmd .= "  echo \"Using sfdisk.\"\n";
$fdiskcmd .= "  sfdisk $device < ../metadata/${outputfilename}.sfd\n";
$fdiskcmd .= "else\n";
$fdiskcmd .= "  echo \"using fdisk.\"\n";
$fdiskcmd .= "  fdisk $geometry $device \< ../metadata/$outputfilename\n";
$fdiskcmd .= "fi\n\nsync\n\n";


print OUTPUT $fdiskcmd;
print OUTPUT $format;

print OUTPUT "fdisk -l $device\n";

close (OUTPUT);

# Now build the script that will build the mount points on the root
# and other partitions.

open (OUTPUT, "> ${outputfilepath}bin/mount.$outputfilename")
    or die "Couldn't open output file ${outputfilepath}bin/make.$outputfilename.\n";

print OUTPUT <<FINIS;
```

```
#! /bin/sh

# A script to create a minimal directory tree on the target hard drive
# and mount the partitions on it. Created at bare metal backup time by
# the Perl script make.fdisk.
FINIS

&gpl (*OUTPUT, "2001");

print OUTPUT <<FINIS;

# WARNING: If your Linux system mount partitions across hard drive
# boundaries, you will have multiple "mount.dev.* scripts. You must
# ensure that they run in the proper order. The root partition should
# be mounted first, then the rest in the order they cascade. If they
# cross mount, you'll have to handle that manually.

FINIS


# We have a hash of mount points and devices in %mountpoints. However,
# we have to process them such that directories are built on the
# appropriate target partition. E.g. where /usr/local is on its own
# partition, we have to mount /usr before we build /usr/local. We can
# ensure this by sorting them. Shorter mount point paths will be built
# first. We can't sort a hash directly, so we use an array.

# We build commands to create the appropriate mount points and then
# mount the partitions to the mount points. This is in preparation for
# untarring the contents of the ZIP disk, done in restore.metadata.

# In case it's all one big partition...
print OUTPUT "mkdir -p $target\n";

foreach $point ( sort keys %mountpoints) {
#     print ("Mount point is $point.\n");
    print OUTPUT "\n# $point is the mountpoint for";
    print OUTPUT " device $mountpoints{$point}.\n";
    print OUTPUT "mkdir -p $target$point\n";
    print OUTPUT "mount $mountpoints{$point} $target$point\n";
}

print OUTPUT "\nmount | grep -i \"/target\"\n";

close (OUTPUT);

# These scripts are dangerous & should only be visible to root.

chmod 0700, "${outputfilepath}bin/make.$outputfilename";
chmod 0700, "${outputfilepath}bin/mount.$outputfilename";
chmod 0600, "${outputfilepath}metadata/${outputfilename}*";
```

11.1.2. `make.gdisk`

This script, run at backup time, creates scripts similar to `make.dev.hda` and `mount.dev.x`, below, for you to run at restore time. The main difference between it and **make.fdisk** is that **make.gdisk** is aimed at GPT devices. It also produces data files similar to `dev.hda` and `dev.hda.sfd`, below. The names of the scripts and data files produced depend on the device given this script as a parameter. Those scripts, run at restore time, build and mount the partitions on the hard drive. `make.gdisk` is called from `save.metadata`, below.

> **Warning**
>
> `make.fdisk`, above, uses the older **fdisk** program, suited to MS-DOS partition tables. When possible and appropriate, use **make.gdisk**, below, instead.

```
#! /usr/bin/perl

# A perl script to create a script and input file for sfdisk to
# re-create the partitions on the hard disk, and format the Linux and
# Linux swap partitions. The first parameter is the fully qualified
# path of the device of the hard disk, e.g. /dev/hda. The two
# resulting files are the script make.dev.x and the data file dev.x
# (where x is the hard drive described, e.g. hda, sdc). make.dev.x is
# run at restore time to rebuild hard drive x, prior to running
# restore.metadata. dev.x is the input file for gdisk.

# The directory tree where everything is put must already exist and be
# specified in the environment variable $zip.

# This program is free software; you can redistribute it and/or modify it
# under the terms of the GNU General Public License as published by the
# Free Software Foundation; either version 2 of the License, or (at your
# option) any later version.

# This program is distributed in the hope that it will be useful, but
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
# General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
# 02110-1301, USA

# You can also contact the Free Software Foundation at
# http://www.fsf.org/

use UUID;                       # for validating uuids.

# It may be necessary to adjust this from time to time as the output
# from sfdisk varies. At least, that is our experience with
# fdisk. This format string is used by unpack to split the output from
# sfdisk into suitable variables for each useful line. We skip the
# size.
```

```
$fmt = 'A4 A17 a15 x15 a4 x2 A*';


# Sub gpl, a subroutine to ship the GPL and other header information
# to the current output file.

sub gpl {
    my $FILE = shift;
    my $year = shift;

    print $FILE <<FINIS;

# Copyright $year through the last date of modification Charles Curley.

# This program is free software; you can redistribute it and/or modify it
# under the terms of the GNU General Public License as published by the
# Free Software Foundation; either version 2 of the License, or (at your
# option) any later version.

# This program is distributed in the hope that it will be useful, but
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
# General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
# 02110-1301, USA

# You can also contact the Free Software Foundation at http://www.fsf.org/

# For more information contact the author, Charles Curley, at
# http://www.charlescurley.com/.

FINIS

}

sub getBootSector {
    my $infile = $_[0];
    my $outfile = $_[1];

    $systemcmd = "dd if=$infile of=$outfile bs=512 count=1 &> /dev/null ";
    system ($systemcmd);
}


# If we have one & only one swap partition, then this must be
# it. Otherwise the user is on her own. We scan fstab for swap mount
# points that have labels for their devices. If there is one and only
# one, we assume that's it, otherwise pass.
```

```
sub getswaplabel {
    my $dev = $_[0];

    open (FSTAB, "< /etc/fstab")
        or die "Couldn't fork: $!\n";
    while (defined (my $line = <FSTAB>)) {
        chomp ($line);
        @fstabs = split (" ", $line);
        if (@fstabs[1] eq "swap") {
            $swaplabel = @fstabs[0];
            if ($swaplabel =~ /LABEL/) {
                $swaps++;
                $sl = substr ($swaplabel, 6);
            }
            #           print ("\"@fstabs[0]\", \"@fstabs[1]\", \"$sl\", $swaps.\n");
            break;
        }
    }
    close (FSTAB);

    # print ("label is $sl.\n");

    if ($swaps == 1) {
        $ret = "-L $sl $dev\n";
    } else {
        $ret = "$dev\n";
    }

    #   print ("Returning :$ret\n");

    return $ret;
}

# doluks, for handing LUKS partitions.

sub doluks {
    my $device = shift;
    $devName=substr ($device, 5);
    $cryptName = "${devName}_crypt";

    $fileName = "${outputfilepath}metadata/$devName-header-backup";
    if ( -e $fileName) {
        unlink ($fileName);
    }
    if (system ("cryptsetup luksHeaderBackup $device --header-backup-file $fileName") != 0 ) {
        die ("LUKS backup of $device failed!\n\n");
    }
    $format .= "echo Device $device is a LUKS partition.\n";
    $format .= "cryptsetup luksHeaderRestore $device --header-backup-file ../metadata/$devName-heade
    $format .= "if ! cryptsetup isLuks $device ; then\n";
    $format .= "    echo Error: $device is NOT a LUKS partition.\n";
    $format .= "    exit\n";
    $format .= "fi\n\n";
```

```
    $format .= "echo Opening $device as a LUKS partition.\n";
    $format .= "cryptsetup luksOpen $device $cryptName\n";
    $format .= "ls /dev/mapper\n";

    print ("\n\nWarning!! $device is a LUKS partition. Be sure you have the passphrase for restorati

    dolvm ();
}


# dolvm is a subroutine to handle LVM partitions.

$lvms = 0;                      # true if we've been here before

sub dolvm {

    #     print ("In dolvm ()...\n");

    if ($lvms == 0) {
        $lvms = 1;

        # Scan /etc/fstab for the logical volumes and write a script to
        # make file systems on them and another to mount 'em later on.

        $mklvs = open (MKLVS, "> ${outputfilepath}bin/make.lvs")
            or die "Couldn't fork: $!\n";

        print MKLVS <<FINIS;
#! /bin/sh

# A script to create file systems on logical volumes. Created at bare
# metal backup time by the Perl script make.gdisk.
FINIS

        &gpl (*MKLVS, "2006");


        print MKLVS <<FINIS;

# Where mke2fs expects to find its configuration file.
export MKE2FS_CONFIG=../mke2fs.conf

export blockcheck=\"\$1\";

if [ "\$blockcheck" != "-c" ] && [ -n "\$blockcheck" ]
then
    echo "\${0}: Build file systems on logical volumes."
    echo "\${0}: -c: block check during file system making."
    exit 1;
fi

FINIS
```

```
        $mtlvs = open (MTLVS, "> ${outputfilepath}bin/mount.lvs")
            or die "Couldn't fork: $!\n";

        print MTLVS <<FINIS;
#! /bin/sh

# A script to mount file systems on logical volumes. Created at bare
# metal backup time by the Perl script make.gdisk.
FINIS

        &gpl (*MTLVS, "2007");


        # Now cycle through all the known logical volumes & set them
        # up. N.B.: This has been tested on machines with only one LV
        # and with two. It picks up both on that test machine.

        $pvdisp = open (PVDISP, "pvdisplay -c |")
            or die ("Can't open LVM display.\n");
        while (defined (my $pv = <PVDISP>)) {
            chomp ($pv);
            #             print ("$pv\n");
            @pv = split (":", $pv);
            $uuid = @pv[11];
            $pvname = @pv[1];
            $phv = @pv[0];
            #             print ("pv $pvname has uuid $uuid.\n");

            # back up the LVM's lvm details. Get the config files.
            if (system ("vgcfgbackup -f ${outputfilepath}metadata/LVM.backs.$pvname $pvname > /dev/n
                die ("Failed to back up $pvname!!\n");
            }

            print (MKLVS "printf \"y\\n\" | pvcreate -ff --uuid \"$uuid\"\\n");
            print (MKLVS "    --restorefile ../metadata/lvm/backup/${pvname} $phv\n");
            print (MKLVS "vgcfgrestore --file ../metadata/LVM.backs.$pvname $pvname\n\n");
        }

        # print (MKLVS "# Hideously disty dependent! turn on LVM.\n");
        # print (MKLVS "if [ -e /etc/init.d/lvm ] ; then\n");
        # print (MKLVS "    /etc/init.d/lvm start\nfi\n\n");
        # print (MKLVS "if [ -e /etc/init.d/lvm2 ] ; then\n");
        # print (MKLVS "    /etc/init.d/lvm2 start\nfi\n\n");

        # Now tell the volume manager software to find the new volume:

        print (MKLVS "# scan for new volume groups\n");
        print (MKLVS "vgscan --mknodes\n\n");
        print (MKLVS "# Activate the volume groups\n");
        print (MKLVS "vgchange -ay\n\n");
        print (MKLVS "ls /dev/mapper\n\n");

        # Now walk fstab in search of logical volumes. This is
```

```
# necessary to pick up swap partitions, and it may pick up
# others. We need fstab below to match the partitions up with
# their mount points, so we keep the array around.

%volsfound = ();
open (FSTAB, "< /etc/fstab")
    or die "Couldn't fork: $!\n";
@fstab = <FSTAB>;
foreach $line (@fstab) {
    @fstabs = split (" ", $line);
    #                   Red Hat|Ubuntu/Debian
    if (@fstabs[0] =~ /VolGroup|mapper/ ) {
        # print ("$line");
        if (@fstabs[2] eq "swap") {
            # Do we need the UUID here?
            print (MKLVS "\necho\necho making LV @fstabs[0] a swap partition.\n");
            print (MKLVS "cmd=\"mkswap \$blockcheck @fstabs[0]\"\n");
            print (MKLVS "echo \"\$cmd\"\n\$cmd\n"); # show it, then execute it.
        } elsif (@fstabs[2] == "ext4") {
            print (MKLVS "echo\necho making LV @fstabs[0], mounted on @fstabs[1],");
            print (MKLVS " an ext4 partition.\n");
            print (MKLVS "cmd=\"mke2fs -t ext4" . getextopts (@fstabs[0]));
            print (MKLVS "\$blockcheck @fstabs[0]\"\n"); # make the command.
            print (MKLVS "echo \"\$cmd\"\n\$cmd\n"); # show it, then execute it.

            print (MTLVS "mkdir -p /target$fstabs[1]\n");
            print (MTLVS "mount @fstabs[0] /target$fstabs[1]\n\n");
            $volsfound{@fstabs[0]} = 4;
        } elsif (@fstabs[2] == "ext3") {
            print (MKLVS "echo\necho making LV @fstabs[0], mounted on @fstabs[1],");
            print (MKLVS " an ext3 partition.\n");
            print (MKLVS "cmd=\"mke2fs -t ext3" . getextopts (@fstabs[0]));
            print (MKLVS "\$blockcheck @fstabs[0]\"\n");
            print (MKLVS "echo \"\$cmd\"\n\$cmd\n"); # show it, then execute it.

            print (MTLVS "mkdir -p /target$fstabs[1]\n");
            print (MTLVS "mount @fstabs[0] /target$fstabs[1]\n\n");
            $volsfound{@fstabs[0]} = 3;
        } elsif (@fstabs[2] == "ext2") {
            print (MKLVS "echo\necho making LV @fstabs[0], mounted on @fstabs[1],");
            print (MKLVS " an ext2 partition.\n");
            print (MKLVS "cmd=\"mke2fs -t ext2 \$blockcheck @fstabs[0]\"\n");
            print (MKLVS "echo \"\$cmd\"\n\$cmd\n"); # show it, then execute it.

            print (MTLVS "mkdir -p /target$fstabs[1]\n");
            print (MTLVS "mount @fstabs[0] /target$fstabs[1]\n\n");
            $volsfound{@fstabs[0]} = 2;
        } else {
            print ("Opps, unknown type of logical volume, @fstabs[0]\n");
        }
    }
}
```

```
print (MKLVS "# End of fstab device list.\n\n");

#         print ("Volumes already found are: ");
#         while ( ($k, $v) = each %volsfound ) {
#             print ("$k ==> $v ");
#         }
#         print ("\n");

# Now walk the logical volume devices and pick up any partitions
# formated ext3/ext2/ext4. This may result in duplicates if the
# partitions have labels but are mounted by device name rather
# than by label.

# FIX ME: This loop doesn't distinguish between ext3 and
# ext4. It may not need to, as we pull the options from the
# partition itself. If so, that would be nice.

opendir (DEVHANDLE, "/dev") or die ("Can't open /dev!!\n");
while ( defined ($fname = readdir (DEVHANDLE))) {
    @fnames = (@fnames, $fname);
}
@sorted = sort (@fnames);
foreach $fname (@sorted) {
    #               Red Hat|Ubuntu
    if ($fname =~ /^VolGroup|^mapper/ && -d "/dev/$fname") {
        # print ("Inside /dev is $fname.\n");
        opendir (VOLHANDLE, "/dev/$fname")
            or die ("Can't open /dev/$fname!!\n");
        while ( defined ($vname = readdir (VOLHANDLE))) {
            @vnames = (@vnames, $vname);
        }
        @vsorted = sort (@vnames);
        foreach $vname (@vsorted) {
            # print ("/dev/$fname/$vname: " . $volsfound{"/dev/$fname/$vname"} . "\n");
            if($vname ne "." && $vname ne ".."
               && $volsfound{"/dev/$fname/$vname"} < 1) {
                #                          print ("Inside /dev/$fname is $vname.\n");
                my $journal = 0;

                # FIX ME: add tests to be sure it's a symlink
                # to a block device.

                # Is it extX?
                if (! system ("dumpe2fs /dev/$fname/$vname 1> /dev/null 2>&1")) {

                    # Is it extX?
                    open (DUMP, "dumpe2fs /dev/$fname/$vname 2> /dev/null|");

                    @lines = <DUMP>;

                    # if it is short, it is most likely an error message.
                    if (scalar (@lines) > 5) {
```

```
                    # If we've gotten here we have a valid
                    # ext[2|3|4] file system. Now prepare to
                    # spit out the commands to recreate
                    # it. Get the label, if any, and whether
                    # there is a journal or not.

                    foreach $_ (@lines) {
                        if (/Filesystem volume name:/) {
                            $label = substr ($_, 26);
                            chop ($label);
                            # print ("\$label is \"$label\".\n");
                          }
                        if (/Filesystem UUID:/) {
                            $uuid = substr ($_, 26);
                            chop ($uuid);
                            # print ("\$uuid is \"$uuid\".\n");
                        }
                        if (/has_journal/) {
                            $journal = 1;
                        }

                    }

                    # Get the mount point, from fstab so
                    # we can mount it.
                    $mountpoint = "none!";
                    foreach $fstab (@fstab) {
                      @fstabs = split (" ", $line);
                      if (@fstabs[0] eq "LABEL=$label" or @fstabs[0] eq "UUID=$uuid") {
                        $mountpoint = @fstabs[1];
                        $type = @fstabs[3];
                        # print ("Mount point is \"$mountpoint\". Type is $type.\n");
                        last;
                      }
                    }
                    if (length ($label) ) {
                      $label = "-L \"" . $label . "\"";
                    }

                    print (MKLVS "echo\necho making LV /dev/$fname/$vname");
                    print (MKLVS " an ext3/4 partition.\n");
                    print (MKLVS "cmd=\"mke2fs -t $type" . getextopts ("/dev/$fname/$vna
                    print (MKLVS "$label \$blockcheck /dev/$fname/$vname\"\n");
                    print (MKLVS "echo \"\$cmd\"\n\$cmd\n"); # show it, then execute it.

                    if ($mountpoint ne "none!") {
                      print (MTLVS "mkdir -p /target$mountpoint\n");
                      print (MTLVS "mount /dev/$fname/$vname /target$mountpoint\n\n");
                    }
                }
            }
        }
    }
```

```
                closedir (VOLHANDLE);
            }
        }
        print (MKLVS "ls /dev/mapper\n");
        print (MTLVS "mount | grep \"/target\"\n");

        closedir (DEVHANDLE);

        close (FSTAB);
        close (MKLVS);
        close (MTLVS);

        chmod 0700, "${outputfilepath}bin/make.lvs";
        chmod 0700, "${outputfilepath}bin/mount.lvs";

        # Copy the LVM configuration to where we can get at it...
        system ("cp -rp /etc/lvm ${outputfilepath}metadata/");

    }

    #     print ("Leaving dolvm ()...\n");

    return ($ret);
}

sub getuuid {
    my $device = shift;

    # print ("In getuuid. \$device = $device.\n");

    my $rs = open (UUID, "blkid -o value -s UUID $device |")
        or die ("Can't open volume id tool.\n");
    while (defined (my $uuid = <UUID>)) {
        chomp ($uuid);
        # print ("$uuid\n");

        if (UUID::parse($uuid, my $rawUuid) != -1) {
            # print ("Device $device has uuid '$uuid'.\n");
            return ($uuid);
        }
    }
    return ('');
}

# scan the partition for the options to mke2fs.

sub getextopts {
    my $device = $_[0];
    # print ("Device is $device\n");

    open (DUMP, "dumpe2fs $device 2> /dev/null|") or
        die ("Can't open a dump.\n");
```

```
    my @lines = <DUMP>;
    if (scalar (@lines) > 1) {

        foreach $_ (@lines) {
            # print ("line: $_");

            if (/Filesystem features/) {
                $_ = substr ($_, 26);
                chomp ();

                # remove features mke2fs on finnix doesn't
                # support. Also needs_recovery, which is superflouous.
                # $_ =~ s/metadata_csum_seed//g;
                # $_ =~ s/metadata_csum//g;
                $_ =~ s/orphan_present//g;
                $_ =~ s/needs_recovery//g;
                $_ = join (',', split (" "));
                # print ("File system features of $device are: |${_}|.\n");
                return ($_ = ' -O ' . $_ . ' ');
            }
        }
        die ("Couldn't get features from device $device\n");
    } else {
        die ("Couldn't run dumpe2fs on $device.\n");
    }
}

# Begin main line code.

$outputfilepath = $ENV{"zip"} . "/";
if ($outputfilepath eq "/") {
    $outputfilepath =  "./";
}
# print "\$outputfilepath is ${outputfilepath}\n";

# These will "fail" silently if the directory is already there, which
# it should be if we're called from save.metadata. These simplify
# testing, though.

mkdir ("${outputfilepath}metadata", 0700);
mkdir ("${outputfilepath}bin", 0700);

# Provide a default device.

# print "\$ARGV[0] is $ARGV[0].\n";

$device = defined ($ARGV[0]) ? $ARGV[0] : "/dev/hda";

# print "Device is $device.\n";

# define fields in the array @_.
$part = 0;
$bootable = 1;
```

```
$firstcyl = 1;
$endcyl = 2;
$parttype = 3;
$desc = 4;

$target = "\/target";

$outputfilename = $device;
$outputfilename =~ s/\//./g;
$outputfilename = substr ($outputfilename, 1, 100);

# Make a hash of the labels.
$mpid = open (MOUNT, "mount -l |") or die "Couldn't fork: $!\n";
while (<MOUNT>) {
    if ($_ =~ /^$device/i) {  # is this a line with a partition in it?
        chomp;
        # print "mount line is \"$_\"\n"; # print it just for grins
        @_ = split;
        if ($_[6] ne "") {     # only process if there actually is a label
            $_[6] =~ s/[\[\]]//g;     # strike [ and ].
            $labels{$_[0]} = $_[6];
            # print "The label of file device $_[0] is $labels{$_[0]}.\n";
        }

        # foreach $el (@_) {
        #     print ("Element is $el\n");
        # }

        # We only mount if it's ext2fs, ext3fs, ext4fs, or EFI, and
        # read and write.

        if ($_[4] =~ /ext[234]/ and $_[5] =~ /\(rw/ ) {
            $type = $_[4];
            # print ("R/W ext[234] partition found.\n");
            if ($_[0] =~ /ide/i) {

                # We have a devfs system, e.g. Mandrake. This code
                # converts back from the devfs designation to the old
                # /dev/hd* designation for tomsrtb. I have NOT checked
                # this out for drives other than /dev/hda. Also, this
                # code does not handle SCSI drives.

                if ( $_[0] =~ /target0/ && $_[0] =~ /bus0/ ) {
                    $letter = 'a';
                } elsif ( $_[0] =~ /target1/ && $_[0] =~ /bus0/) {
                    $letter = 'b';
                } elsif ( $_[0] =~ /target0/ && $_[0] =~ /bus1/) {
                    $letter = 'c';
                } else {
                    $letter = 'd';
                }
                $_[0] =~ s|/ide/host\d+/bus\d+/target\d+/lun\d+/part|/hd|g;
                $_[0] =~ s/hd/hd$letter/;
```

```
            }
            $mountpoints{$_[2]} = $_[0];
            # print "$_[2] is the mountpoint for finnix";
            # print " device $mountpoints{$_[2]}.\n";
          } elsif ($_[2] eq '/boot/efi' and $_[4] =~ /vfat/ and $_[5] =~ /\(rw/ ) {
            print ("R/W EFI partition found.\n");
            $mountpoints{$_[2]} = $_[0];
            # print "$_[2] is the mountpoint for finnix";
            # print " device $mountpoints{$_[2]}.\n";
          }
    }
}
close (MOUNT);

# Get sfdisk output. If we have sfdisk at restore time (e.g. Knoppix),
# we'll use it.

system "sfdisk --dump $device > ${outputfilepath}metadata/${outputfilename}.sfd";

# Otherwise we'll use the output from gdisk, which may or may not be
# any more accurate.

$fpid = open (GDISK, "gdisk -l $device |") or die "Couldn't fork: $!\n";

open (OUTPUT, "> ${outputfilepath}metadata/${outputfilename}")
    or die "Couldn't open output file ${outputfilepath}metadata/${outputfilename}.\n";

while (<GDISK>) {
    if ($_ =~ /^   /i) { # is this the line with our partition in it?
        $bootFlag = 0;

        # print "Line: $_";       # print it just for grins
        # print "      0        10        20        30        40        50        60\n";
        chomp;                   # kill trailing \r

        @_ = unpack ($fmt, $_);

        # Now strip white spaces from cylinder numbers, white space &
        # leading plus signs from partition type.
        @_[$part] =~ s/[ \t]+//;
        @_[$firstcyl] =~ s/[ \t]+//;
        @_[$endcyl] =~ s/[ \t]+//;
        @_[$parttype] =~ s/[+ \t]+//;

        $partnumber = "@_[$part]";
        if ($device =~ /nvme/i) {
          $partprefix = 'p';
        } else {
          $partprefix = ''
        }
        $partition = $device . $partprefix . $partnumber;

        # # just for grins
```

```
# print "Partition: |$partnumber|, Start: |@_[$firstcyl]|, Stop: |@_[$endcyl]|,";
# print " Type: |@_[$parttype]|, description is |@_[$desc]|, partition is $partition on devi

# foreach $el (@_) {
#     print ("Element is $el\n");
# }

# Here we start creating the input to recreate the partition
# this line represents.

print OUTPUT "n\n";

if (system ("cryptsetup isLuks $partition") == 0) {
    print ("$partition is a LUKS partition! Woo Hoo!!\n");
    doluks ($partition);
} else {
    if (@_[$parttype] eq "8300" ) {
        # Linux partition
        print OUTPUT "l\n@_[$firstcyl]\n";
        if (@_[$firstcyl] ne @_[$endcyl]) {
            print OUTPUT "@_[$endcyl]\n";
        }
        if ($labels{$partition}) { # do we have a label?
            $format .= "echo\necho formatting $checking$partition\n";
            $format .= "cmd=\"mke2fs -t $type" . getextopts ($partition) . "\$blockcheck";
            $format .= " -L $labels{$partition} $partition\"\n";
        } else {
            $format .= "echo\necho formatting $checking$partition\n";
            $format .= "cmd=\"mke2fs -t $type" . getextopts ($partition) . "\$blockcheck $pa
        }
        $format .= "echo \"\$cmd\"\n\$cmd\n"; # show it, then execute it.
        $uuid = getuuid ($partition);
        if (length ($uuid) > 0) {
            $format .= "tune2fs -U $uuid $partition\n";
        }

      } elsif ( @_[$parttype] eq "EF00" ) {
        # EFI partition. Note: we use extremely high
        # compression, which takes time and memory. You can
        # adjust that with the parameter following the -Z in
        # the next line below.
        if (system ("fsarchiver -oAaZ 21 savefs ${outputfilepath}metadata/$outputfilename$pa
          die ("fsarchiver failed! Is it installed?\n\n")
        }
        $format .= "echo Restoring EFI partition $partition.\n";
        $format .= "fsarchiver restfs ../metadata/$outputfilename${partprefix}${partnumber}.

      } elsif ( @_[$parttype] eq "8200" ) {
        # Linux swap. Does not do labels. Yet. It's the UUID
        # we want, not the PARTUUID.
        my $uuid = getuuid ($partition);
        print "\$uuid is $uuid.\n";
        if (length ($uuid) > 0) {
```

```
                $uuid = '-uuid ' . $uuid;
              }
              $format .= "\necho\necho Making $partition a swap partition.\n";
              $format .= "cmd=\"mkswap \$blockcheck $uuid $partition\"\n";
              $format .= "echo \"\$cmd\"\n\$cmd\n"; # show it, then execute it.

            # } elsif ( 1 == 2 ) { # i.e. never. Do we want to preserve other DOS partitions?
            #   # || @_[$parttype] == 4 || @_[$parttype] == 6 ||
            #   # @_[$parttype] eq "b" || @_[$parttype] eq "c" ||
            #   # @_[$parttype] eq "e" || @_[$parttype] eq "12" ) {
            #   # Non-EFI dos partition. We don't handle hidden
            #   # partitions.
            #   print ("Making DOS partition.\n");

            #   getBootSector ($partition, "${outputfilepath}metadata/$outputfilename$partprefix$p

            #   print OUTPUT "l\n$partnumber\n@_[$firstcyl]\n";
            #   # in case it's all on one cylinder
            #   if (@_[$firstcyl] ne @_[$endcyl]) {
            #       print OUTPUT "@_[$endcyl]\n";
            #   }
            #   $typechanges .= "t\n$partnumber\n@_[$parttype]\n";
            #   $format .= "echo\necho formatting $checking$partition\n";
            #   $format .= "mkdosfs \"\$blockcheck\"";
            #   if ( @_[$parttype] eq "EF00" || @_[$parttype] == c
            #        || @_[$parttype] eq "12" ) {
            #       # We have a FAT32 partition. Add a command line switch.
            #       $format .= " -F 32";
            #   }
            #   $format .= " $partition\n";
            #   $format .= "# restore FAT boot sector.\n";
            #   $format .= "dd if=$outputfilename$partprefix$partnumber";
            #   $format .= " of=$partition bs=512 count=1\n\n";
            } elsif ( @_[$parttype] eq "8E00" ) {
                dolvm ();
              } else {
                print ("Ignoring partition $partition, type @_[$parttype], description @_[$desc].\n"
              }

            $format .= "\n";
        }
    }
}

# Append all the partition type changes, validate, and print out the
# results.

print OUTPUT "${typechanges}v\nw\n";

close (OUTPUT);
close (GDISK);
```

```
open (MAKEOUTPUT, "> ${outputfilepath}bin/make.$outputfilename")
    or die "Couldn't open output file ${outputfilepath}bin/make.$outputfilename.\n";

print MAKEOUTPUT <<FINIS;
#! /bin/sh

# A script to restore the partition data of a hard drive and format
# the partitions. Created at bare metal backup time by the Perl script
# make.gdisk.
FINIS

&gpl (*MAKEOUTPUT, "2001");

print MAKEOUTPUT <<FINIS;

swapoff -a

# # Hideously disty dependent! Turn off LVM.
# if [ -e /etc/init.d/lvm ] ; then
#     /etc/init.d/lvm stop
# fi
# if [ -e /etc/init.d/lvm2 ] ; then
#     /etc/init.d/lvm2 stop
# fi

export blockcheck=\"\$1\";

if [ "\$blockcheck" != "-c" ] && [ -n "\$blockcheck" ]
then
    echo "\${0}: automated restore with no human interaction."
    echo "\${0}: -c: block check during file system making."
    exit 1;
fi

FINIS

# Clean the old partition table out.
print MAKEOUTPUT "echo Zeroing out the hard drive. This may take a while.\n";
print MAKEOUTPUT "dd if=/dev/zero of=$device bs=1024 count=1000000 status=progress\n";
print MAKEOUTPUT "sync\n\n";


# command for sfdisk/fdisk

$fdiskcmd .= "# see if we have sfdisk & if so use it.\n";
$fdiskcmd .= "if which sfdisk ; then\n";
$fdiskcmd .= "  echo \"Using sfdisk.\"\n";
$fdiskcmd .= "  sfdisk $device < ../metadata/${outputfilename}.sfd\n";
$fdiskcmd .= "else\n";
$fdiskcmd .= "  echo \"using fdisk.\"\n";
$fdiskcmd .= "  fdisk $geometry $device \< ../metadata/$outputfilename\n";
$fdiskcmd .= "fi\n\nsync\n\n";
```

```
print MAKEOUTPUT $fdiskcmd;

print MAKEOUTPUT "# Where mke2fs expects to find its configuration file.\n";
print MAKEOUTPUT "export MKE2FS_CONFIG=../mke2fs.conf\n\n";


print MAKEOUTPUT $format;

print MAKEOUTPUT "gdisk -l $device\n";

close (MAKEOUTPUT);

# Now build the script that will build the mount points on the root
# and other partitions.

open (MOUNTOUTPUT, "> ${outputfilepath}bin/mount.$outputfilename")
    or die "Couldn't open output file ${outputfilepath}bin/make.$outputfilename.\n";

print MOUNTOUTPUT <<FINIS;
#! /bin/sh

# A script to create a minimal directory tree on the target hard drive
# and mount the partitions on it. Created at bare metal backup time by
# the Perl script make.gdisk.
FINIS

&gpl (*MOUNTOUTPUT, "2001");

print MOUNTOUTPUT <<FINIS;

# WARNING: If your Linux system mount partitions across hard drive
# boundaries, you will have multiple "mount.dev.* scripts. You must
# ensure that they run in the proper order. The root partition should
# be mounted first, then the rest in the order they cascade. If they
# cross mount, you'll have to handle that manually.

FINIS


# We have a hash of mount points and devices in %mountpoints. However,
# we have to process them such that directories are built on the
# appropriate target partition. E.g. where /usr/local is on its own
# partition, we have to mount /usr before we build /usr/local. We can
# ensure this by sorting them. Shorter mount point paths will be built
# first. We can't sort a hash directly, so we use an array.

# We build commands to create the appropriate mount points and then
# mount the partitions to the mount points. This is in preparation for
# untarring the contents of the ZIP disk, done in restore.metadata.

# In case it's all one big partition...
print MOUNTOUTPUT "mkdir -p $target\n";
```

```
foreach $point ( sort keys %mountpoints) {
    #     print ("Mount point is $point.\n");
    print MOUNTOUTPUT "\n# $point is the mountpoint for";
    print MOUNTOUTPUT " device $mountpoints{$point}.\n";
    print MOUNTOUTPUT "mkdir -p $target$point\n";
    print MOUNTOUTPUT "mount $mountpoints{$point} $target$point\n";
}

print MOUNTOUTPUT "\nmount | grep \"/target\"\n";

close (MOUNTOUTPUT);

# These scripts are dangerous & should only be visible to root.

chmod 0700, "${outputfilepath}bin/make.$outputfilename";
chmod 0700, "${outputfilepath}bin/mount.$outputfilename";
chmod 0600, "${outputfilepath}metadata/${outputfilename}*";
```

11.1.3. `make.dev.hda`

This script is a sample of the sort produced by `make.fdisk`, above. It uses data files like `dev.hda`, below. It builds partitions and puts file systems on some of them. This is the first script run at restore time.

If you are brave enough to edit `dev.hda` or `dev.hda.sfd` (q.v.), say, to add a new partition, you may need to edit this script as well.

If you want make.dev.hda to check for bad blocks when it puts a file system on the partitions, use a "-c" command line option.

```
#! /bin/sh

# A script to restore the partition data of a hard drive and format
# the partitions. Created at bare metal backup time by the Perl script
# make.fdisk.

# Copyright 2001 through the last date of modification Charles Curley.

# This program is free software; you can redistribute it and/or modify it
# under the terms of the GNU General Public License as published by the
# Free Software Foundation; either version 2 of the License, or (at your
# option) any later version.

# This program is distributed in the hope that it will be useful, but
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
# General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
# 02110-1301, USA
```

```
# You can also contact the Free Software Foundation at http://www.fsf.org/

# For more information contact the author, Charles Curley, at
# http://www.charlescurley.com/.


export blockcheck=$1;

if [ "$blockcheck" != "-c" ] && [ -n "$blockcheck" ]
then
    echo "${0}: automated restore with no human interaction."
    echo "${0}: -c: block check during file system making."
    exit 1;
fi

dd if=/dev/zero of=/dev/hda bs=512 count=2

swapoff -a
sync

# see if we have sfdisk & if so use it.
if which sfdisk ; then
  echo "Using sfdisk."
  sfdisk  -H 128 -S 63 -C 523 /dev/hda < dev.hda.sfd
else
  echo "using fdisk."
  fdisk  -H 128 -S 63 -C 523 /dev/hda < dev.hda
fi

sync

echo
echo formatting /dev/hda1
mkdosfs $blockcheck /dev/hda1
# restore FAT boot sector.
dd if=dev.hda1 of=/dev/hda1 bs=512 count=1

echo
echo formatting /dev/hda2
mke2fs -j $blockcheck -L /boot /dev/hda2

echo
echo formatting /dev/hda3
mke2fs -j $blockcheck -L / /dev/hda3

echo Making /dev/hda5 a swap partition.
mkswap $blockcheck /dev/hda5

fdisk -l "/dev/hda"
```

11.1.4. make.lvs

make.lvs is generated by make.fdisk, but only if logical volumes are present. As the name suggests, it builds the logical volumes and makes file systems on them.

```
#! /bin/sh

# A script to create file systems on logical volumes. Created at bare
# metal backup time by the Perl script make.fdisk.

# Copyright 2006 through the last date of modification Charles Curley.

# This program is free software; you can redistribute it and/or modify it
# under the terms of the GNU General Public License as published by the
# Free Software Foundation; either version 2 of the License, or (at your
# option) any later version.

# This program is distributed in the hope that it will be useful, but
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
# General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
# 02110-1301, USA

# You can also contact the Free Software Foundation at http://www.fsf.org/

# For more information contact the author, Charles Curley, at
# http://www.charlescurley.com/.


export blockcheck=$1;

if [ "$blockcheck" != "-c" ] && [ -n "$blockcheck" ]
then
    echo "${0}: Build file systems on logical volumes."
    echo "${0}: -c: block check during file system making."
    exit 1;
fi

export LVM_SYSTEM_DIR=$(pwd)/lvm.cfg

echo "y\n" | pvcreate -ff --uuid "CCmw0N-0We2-HzRS-jRZa-FkC7-NxTc-oAfvpX"\
     --restorefile lvm.cfg/archive/VolGroup00_*.vg   /dev/hda3
vgcfgrestore --file LVM.backs VolGroup00

# Hideously disty dependent!
if [ -e /etc/init.d/lvm ] ; then
     /etc/init.d/lvm start
fi
```

```
echo
echo making LV /dev/VolGroup00/LogVol00 an ext3 partition.
mke2fs -j $blockcheck /dev/VolGroup00/LogVol00

echo
echo making LV /dev/VolGroup00/LogVol02 an ext3 partition.
mke2fs -j $blockcheck /dev/VolGroup00/LogVol02

echo
echo making LV /dev/VolGroup00/LogVol01 a swap partition.
mkswap $blockcheck /dev/VolGroup00/LogVol01
```

11.1.5. `mount.dev.hda`

This script is a sample of the sort produced by `make.fdisk`, above. It builds mount points and mounts partitions on them, making the target file system ready for restoring files. This is the second script run at restore time.

If you are brave enough to edit `dev.hda` (q.v.), say, to add a new partition, you may need to edit this script as well.

```
#! /bin/sh

# A script to create a minimal directory tree on the target hard drive
# and mount the partitions on it. Created at bare metal backup time by
# the Perl script make.fdisk.

# Copyright 2001 through the last date of modification Charles Curley.

# This program is free software; you can redistribute it and/or modify it
# under the terms of the GNU General Public License as published by the
# Free Software Foundation; either version 2 of the License, or (at your
# option) any later version.

# This program is distributed in the hope that it will be useful, but
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
# General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
# 02110-1301, USA

# You can also contact the Free Software Foundation at http://www.fsf.org/

# For more information contact the author, Charles Curley, at
# http://www.charlescurley.com/.


# WARNING: If your Linux system mount partitions across hard drive
# boundaries, you will have multiple "mount.dev.* scripts. You must
# ensure that they run in the proper order. The root partition should
# be mounted first, then the rest in the order they cascade. If they
```

```
# cross mount, you'll have to handle that manually.


# / is the mountpoint for tomsrtbt device /dev/hda3.
mkdir /target/
mount /dev/hda3 /target/

# /boot is the mountpoint for tomsrtbt device /dev/hda2.
mkdir /target/boot
mount /dev/hda2 /target/boot

mount | grep -i "/dev/hda"
```

11.1.6. `mount.lvs`

`mount.lvs` is generated by `make.fdisk`, but only if logical volumes are present. As the name suggests, it mounts the logical volumes ready for restoration.

```
#! /bin/sh

# A script to mount file systems on logical volumes. Created at bare
# metal backup time by the Perl script make.fdisk.

# Copyright 2006 through the last date of modification Charles Curley.

# This program is free software; you can redistribute it and/or modify it
# under the terms of the GNU General Public License as published by the
# Free Software Foundation; either version 2 of the License, or (at your
# option) any later version.

# This program is distributed in the hope that it will be useful, but
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
# General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
# 02110-1301, USA

# You can also contact the Free Software Foundation at http://www.fsf.org/

# For more information contact the author, Charles Curley, at
# http://www.charlescurley.com/.

mkdir -p /target/
mount /dev/VolGroup00/LogVol00 /target/

mkdir -p /target/home
mount /dev/VolGroup00/LogVol02 /target/home
```

```
mount | grep -i "/target"
```

11.1.7. dev.hda

This data file is used at restore time if **sfdisk** is not present on the restoration Linux. It is fed to **fdisk** by the script `make.dev.hda`. It is produced at backup time by `make.fdisk`. Those familiar with **fdisk** will recognize that each line is an **fdisk** command or value, such as a cylinder number. Thus, it is possible to change the partition sizes and add new partitions by editing this file. That's why the penultimate command is **v**, to verify the partition table before it is written.

```
n
p
1
1
29
a
1
n
p
2
30
44
n
e
3
45
1023
n
l
45
944
n
l
945
1023
t
1
6
t
6
82
v
w
```

11.1.8. dev.hda.sfd

This data file is used at restore time if **sfdisk** is present on the restoration Linux system. It is fed to **sfdisk** by the script `make.dev.hda`. It is produced at backup time by `make.fdisk`. Each line represents a partition. Thus, it is possible to change the partition sizes and add new partitions by editing this file.

```
# partition table of /dev/hda
unit: sectors

/dev/hda1 : start=       63, size=   116865, Id= 6, bootable
/dev/hda2 : start=   116928, size=   153216, Id=83
/dev/hda3 : start=   270144, size=   286272, Id=82
/dev/hda4 : start=   556416, size=  3568320, Id= 5
/dev/hda5 : start=   556479, size=  3568257, Id=83
```

11.1.9. `save.metadata`

This is the first script to run as part of the backup process. It calls `make.fdisk`, above. If you have a SCSI hard drive or multiple hard drives to back up, edit the call to `make.fdisk` appropriately.

> **WARNING:** Recent kernels have incorporated a new ATA (IDE) hard drive driver, libata. Because of this, parallel ATA (PATA) drives now show up as SCSI drives, as serial ATA (SATA) have always done. However, not all rescue distributions (e.g. Finnix) use this new driver. There is a line toward the bottom of `save.metadata` wich very carefully replaces "/dev/sda" with "/dev/hda". Use this as a template if you have multiple IDE hard drives. Comment it out or delete it if this is not an issue for you.
>
> Note that there is no guaranteed mapping! Systems with multiple hard drives may have confusing mappings. Be sure to edit this line carefully. Check it if you add or remove a hard drive of any interface type to or from your system!

```
#! /bin/bash

# A script to save certain meta-data off to the boot partition. Useful for
# restoration.

# Copyright 2000 through the last date of modification, Charles Curley.

# This program is free software; you can redistribute it and/or modify it
# under the terms of the GNU General Public License as published by the
# Free Software Foundation; either version 2 of the License, or (at your
# option) any later version.

# This program is distributed in the hope that it will be useful, but
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
# General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
# 02110-1301, USA

# You can also contact the Free Software Foundation at
# http://www.fsf.org/

# 2013-06-14: Preliminary work on preserving GPT information. We have
# not automated restoring, but at least we preserve the partition
```

```
# data.

# 2013-05-09: We now accumulate LSB release data into the README.txt
# file.

# 2012-07-18: Added creation of a list of packages, for debian systems
# only.

# 2007-10-29: added support for Debian systems.

# 2007-05-22: Changes for FHS compliance. Removed commented out
# references to ZIP drives. Added a line to deal with the fact that
# libata (in newer kernels) maps IDE drives to SCSI device names, but
# not all rescue distributions use libata. So we have to change the
# device names from SCSI to IDE, e.g. /dev/sda to /dev/hda.

# 2006-03-26: had a deprecated option in the sort options; fixed that.

# 2005-09-09: Added a line to create a boot disk ISO in the ZIP drive.

# 2005-08-30: Modernized sub-shell calls, a few other tweaks.

# 2005-07-29: Fedora Core 4 mods. Name of the directory to be saved
# has to be last. Also, we now specify --numeric-owner so as to avoid
# UID problems when using some live CD systems. And we now save to
# /var instead of a mounted ZIP disk.

# 2005-02-19: Fedora Core 3 mods.

# 2003 01 08: We now age the output from rpm -VA to make back
# comparisons easier.

# The loop that creates directories now has the -p option for mkdir,
# which means you can create parents on the fly if they don't already
# exist.

# initrd is now in the list of directories to create automatically.

# We now exclude more stuff when building the tarballs.

# 2002 07 01: Went to bzip2 to compress the archives, for smaller
# results. This is important in a 100MB ZIP disk. Also some general
# code cleanup.

# 2002 07 01: The function crunch will tar and bzip2 the
# archives. This is cleaner than the old code, and has better safety
# checking.


# For more information contact the author, Charles Curley, at
# http://www.charlescurley.com/.

# Exclude: given a fully qualified path to an ambiguous file name,
```

```
# expand it to the base name plus, e.g. any version numbering in the
# base name. E.g. "/usr/lib/python*" becomes "python2.5" if that's
# what's in the directory. We then prepend "--exclude" and return
# it. Use it to prepare ambiguous excludes for crunch's benefit. If
# the file doesn't exist, we return nothing.

function exclude {

if [ -z "$1" ]; then            # 0 length parameter check.
   echo "-Parameter #1 is missing.-" # Also if no parameter is passed.
   exit 1
else
   local file=$1
   local t
   for t in ${file} ; do
     if [ -d "${t}" ]; then
       echo -n '--exclude ' "${t#/}" ' ' ;
     fi
   done
fi
}


# Crunch: A function to compress the contents of a directory and put
# the archive onto the ZIP disk.

# The first parameter is the name of the archive file to be
# created. The backup location, $zip, will be prepended and the
# extension, "tar.bz2" will be appended.

# All following parameters will be taken as additional directories or
# files to be put into the archive.

function crunch {
if [ -z "$1" ] || [ -z "$2" ] # Checks if parameter #1 or #2 is zero length.
then
   echo "-Parameter #1 or #2 is missing.-"  # Also if no parameter is passed.
   return 1
else
   local file=$1  # The archive file to create
   shift   # Discard the file name
   local dirs=$*  # The director[y|ies] to archive

   # shall we use multi-processor compression? Assuming it's in our
   # $PATH.
   local zipper
   zipper=$(command -v pbzip2)

   # Do not cross file system boundaries.
   local opts="--numeric-owner --one-file-system -cf"
   if [ -z "${zipper}" ] ; then
       local tarcmd="tar -j $opts" # The tar command.
   else
```

```
        local tarcmd="tar -I ${zipper} $opts" # The tar command.
    fi

    local tarit="$tarcmd  ${zip}/${data}/$file.tar.bz2 $dirs"
    echo "${tarit}"
    $tarit   # do it!!

    local error=$?  # Preserve the exit code

    if [ $error != 0 ]  # Did we fail?
    then    # Yes
       echo "Tar failed with error $error"
       echo "${tarcmd}" "${zip}/${data}/$file.tar.bz2" "${dirs}"
       exit $error  # return tar's exit code as ours
    fi

    return 0   # For error testing if needed.
fi
}

# Begin the main line code
export data="data";             # Name of the data directory in the archive
export today;
if ! today=$(date +%Y%m%d); then  # Today's date
    echo Error getting date "${today}".
    exit 0
fi

export zips="/var/lib/bare.metal.recovery"
export zip="${zips}/${HOSTNAME}.${today}";  # Today's archive

# shellcheck disable=SC2086
if [ -d "${zip}" ] ; then
    rms="${zip}*"               # get the checksums, too.
    echo "Deleting " ${rms} "from earlier today,"
    echo
    rm -r ${rms}
fi

mkdir -p "${zip}"/metadata "${zip}"/bin "${zip}"/${data}

NEW="${zip}"/metadata/rpmVa.txt       # name for the rpm -Va output file.

# Now we save hard drive information. Run make.fdisk on each hard
# drive in the order in which it mounted from the root partition. That
# is, run it first on the hard drive with your root partition, then
# any hard drives that mount to the first hard drive, then any hard
# drives that mount to those. For example, if your root partition is
# on /dev/sdc, run "make.fdisk /dev/sdc" first.

# The reason for this is that make.fdisk produces a script to make
# mount points and then mount the appropriate partition to them during
# first stage restore. Mount points must be created on the partition
```

```
# where they will reside. The partitions must be mounted in this
# order. For example, if your /var and /var/ftp are both separate
# partitions, then you must mount /, create /var, then mount /var,
# then create /var/ftp. The order in which the script "first.stage"
# runs the mounting scripts is based on their time of creation.

# If necessary, put a line, "sleep 1" between calls to make.fdisk.

echo "Saving hard drive info"

# List all your hard drives here. Put them in the order you want
# things done at restore time. N.B. The nvme search pattern will only
# find names with single digits in the first two numeric fields. This
# might be problematic. e.g. it will not find nvme23n1.

for drive in $(find /dev/ -maxdepth 1 \( -name '[hsv]d?' -o -name 'nvme[0-9]n[0-9]' \) | cut -c6-) ;
    echo Processing drive "${drive}"
    if fdisk -l "/dev/${drive}" | grep -qi gpt ; then
        echo "${drive}: GPT found."

        sgd=$(command -v sgdisk)
        if [ $? = '1' ] ; then
            echo "sgdisk required ; install package gdisk!"
            exit 1
        fi

 ${sgd} -p "/dev/${drive}" > "${zip}/sgdisk.${drive}.txt"

        # For use with "sgdisk --load-backup=file"
 ${sgd} --backup="${zip}/metadata/sgdisk.${drive}" "/dev/${drive}"

        make.gdisk "/dev/${drive}"
    else
        echo "${drive}: MBR found."
        make.fdisk "/dev/${drive}"
    fi
    fdisk -l "/dev/${drive}" > "${zip}/fdisk.${drive}"
done

echo -e "$(hostname) bare metal archive, created $(date)" > "${zip}"/README.txt
uname -a >> "${zip}"/README.txt

# More information about disks. Possibly useful for reconstructing
# complicated setups we would otherwise miss.
lsblk -o +size,partflags,hotplug -Mfi  >> "${zip}"/lsblk.Mfi.txt
lsblk -o +size,partflags,hotplug -MfiJ >> "${zip}"/lsblk.MfiJ.txt
lsblk -o +size,partflags,hotplug -MfiP >> "${zip}"/lsblk.pairs.txt

# Preserve the release information. Tested with Red Hat/Fedora, should
# work with SuSE, Mandrake and other RPM based systems. Also Ubuntu
# 7.10, Gutsy Gibbon.

for releasefile in /etc/*release*  /etc/debian_version ; do
```

```
  # echo "${releasefile}"
  if [ -e "${releasefile}" ] && [ ! -L "${releasefile}" ] ; then
    cat "${releasefile}" >> "${zip}/README.txt"
  fi
done

# Linux Standard Base information.
if lsb=$(command -v lsb_release) ; then
# if [ $? = '0' ] ; then
  { echo ;
  echo LSB data:
  ${lsb} -a ; } >> "${zip}"/README.txt
fi

# end Linux Standard Base information.

# Are we on an RPM based system? Suse?? Other RPM systems?
REDHAT=$(grep -iq \(fedora\|redhat\) "${zip}"/README.txt ; echo $?)

ifconfig > "${zip}"/ifconfig

if [ "${REDHAT}" == 0 ] ; then

    # back up RPM metadata
    echo "Verifying RPMs."
    rpm -Va | sort -t ' ' -k 3 | uniq > "${NEW}"
    echo "Finished verifying RPMs."

else
    echo "Non-RPM System."
    # apt-get clean
    # apt-get --yes autoremove
    dpkg --get-selections "*"  > "${zip}/dpk.selections.txt"
    dpkg-query --showformat='${Package}\t${Version}\t${Revision}\t${Architecture}\n' --show | sort >
    if [ -z "$(command -v debsums)" ] ; then
        echo "debsums is *not* intalled."
    else
        echo Running debsums...
        debsums -csa > "${zip}"/metadata/debsums.txt 2>&1 &
    fi
fi

echo "Building the ZIP drive backups."

# These are in case we need to refer to them while rebuilding. The
# rebuilding process should be mostly automated, but you never
# know....

ls -al /mnt > "${zip}"/ls.mnt.txt
ls -al / > "${zip}"/ls.root.txt
ls -al /var > "${zip}"/ls.var.txt

cd / || exit
```

```
cp -r etc/mke2fs.conf "${zip}"/

# Build our minimal archives on the ZIP disk. These appear to be
# required so we can restore later on.

crunch usr.lib \
$(exclude 'usr/lib/anaconda*') \
$(exclude 'usr/lib/evolution*') \
$(exclude 'usr/lib/firefox*') \
$(exclude 'usr/lib/gimp*') \
$(exclude 'usr/lib/ImageMagick*') \
$(exclude 'usr/lib/libgucharmap*' ) \
$(exclude 'usr/lib/libkatepartinterfaces*') \
$(exclude 'usr/lib/libkdeui*' ) \
$(exclude 'usr/lib/libkhtml*') \
$(exclude 'usr/lib/libmwaw*' ) \
$(exclude 'usr/lib/llvm*' ) \
$(exclude 'usr/lib/perl*') \
$(exclude 'usr/lib/python*') \
$(exclude 'usr/lib/qt*') \
$(exclude 'usr/lib/x86_64-linux-gnu/perl*' ) \
--exclude usr/lib/Adobe \
--exclude usr/lib/apache2 \
--exclude usr/lib/calibre \
--exclude usr/lib/chromium \
--exclude usr/lib/cups \
--exclude usr/lib/debug \
--exclude usr/lib/dri \
--exclude usr/lib/gcc \
--exclude usr/lib/gconv \
--exclude usr/lib/git-core \
--exclude usr/lib/goffice \
--exclude usr/lib/googleearth \
--exclude usr/lib/i386-linux-gnu \
--exclude usr/lib/isdn \
--exclude usr/lib/jvm \
--exclude usr/lib/kde4 \
--exclude usr/lib/lapack \
--exclude usr/lib/libreoffice  \
--exclude usr/lib/libvirt \
--exclude usr/lib/openoffice \
--exclude usr/lib/p7zip \
--exclude usr/lib/ruby \
--exclude usr/lib/scons \
--exclude usr/lib/vlc \
--exclude usr/lib/wine \
--exclude usr/lib/X11 \
--exclude usr/lib/xorg \
usr/lib &

# locale
```

```
crunch usr.share \
$(exclude 'fonts*') \
--exclude AAVMF \
--exclude doc \
--exclude foomatic \
--exclude gnome \
--exclude gnome-applets \
--exclude icons \
--exclude man \
--exclude pixmaps \
--exclude selinux \
--exclude texlive \
--exclude X11 \
usr/share &

# crunch usr.share.locale.en_US usr/share/locale/en_US/

# if [ -e /usr/share/fonts/default ]\
#   && [ -e /usr/share/fonts/ISO8859-2 ]\
#   && [ -e /usr/share/fonts/bitmap-fonts ]; then
#     crunch usr.share.fonts /usr/share/fonts/default /usr/share/fonts/ISO8859-2 \
#     /usr/share/fonts/bitmap-fonts
# fi

crunch root --exclude root/tmp --exclude root/.cpan --exclude root/.mozilla --exclude root/down root
crunch boot boot &
crunch etc --exclude etc/samba --exclude etc/gconf etc & #  --exclude etc/X11

# lib64 is now a symlink to usr/lib64. Also, do similarly for other
# architectures, e.g. lib32 & usr/lib32. I suspect this is part of the
# usr merge effort.
if [ -L lib64 ] ; then
    crunch lib64 lib64 &
fi
if [ -d usr/lib64 ] ; then
    crunch usr.lib64 usr/lib64 &
fi

crunch lib lib &

crunch usr.sbin usr/sbin &
crunch usr.local usr/local &
# crunch usr.libexec usr/libexec &

if [ "${REDHAT}" == 0 ] && [ -d usr/kerberos ] ; then
  crunch usr.kerberos usr/kerberos &
fi

crunch usr.bin --exclude usr/bin/emacs-x \
--exclude usr/bin/emacsclient --exclude usr/bin/emacs-nox --exclude \
usr/bin/gs --exclude usr/bin/pine "$(exclude 'usr/bin/gimp-*')" \
--exclude usr/bin/doxygen --exclude usr/bin/postgres --exclude \
usr/bin/gdb --exclude usr/bin/kmail --exclude usr/bin/splint \
```

```
--exclude usr/bin/odbctest --exclude usr/bin/php --exclude \
usr/bin/xchat --exclude usr/bin/gnucash --exclude usr/bin/pdfetex \
--exclude usr/bin/pdftex --exclude usr/bin/smbcacls \
--exclude usr/bin/evolution-calendar --exclude usr/bin/xpdf \
--exclude usr/bin/xmms usr/bin &

crunch sbin sbin &
crunch bin bin &
crunch dev dev &
crunch run --exclude run/user/*/doc run &

crunch var.backup var/backups &

# Debian amanda clients will want these in order to run amrecover:
if [ -d var/lib/amanda ] && [ -d var/log/amanda ] ; then
    crunch var.log.amanda var/log/amanda &
    crunch var.lib.amanda var/lib/amanda &
fi

# How well will this work if you don't run this script after every
# backup? I suspect these will get out of synch with the amanda
# backups.

# End amanda on Debian


# RH8. Fedora 1 puts them in /lib
# crunch kerberos usr/kerberos/lib/ &

# Now optional saves.

# arkeia specific:
# crunch arkeia usr/knox &

# save these so we can use ssh for restore. *crack* for RH 7.0 login
# authentication.
# RH 8.0
# crunch usr.lib usr/lib/*crack* usr/lib/libz* usr/lib/libssl* usr/lib/libcrypto* &
# Fedora 1
# crunch usr.lib usr/lib/*crack* usr/lib/libz* usr/lib/libwrap*\
#  usr/lib/libk* usr/lib/*krb5* /usr/lib/libgss* &
# Fedora 3
# crunch usr.lib usr/lib/*crack* usr/lib/libz* usr/lib/libwrap*\
#  usr/lib/libk* usr/lib/*krb5* usr/lib/libgss* &
# Fedora 7
# crunch usr.lib usr/lib/*crack*\
#  usr/lib/libk* usr/lib/*krb5* usr/lib/libgss* &

# Grub requires these at installation time.
if [ "${REDHAT}" == 0 ] ; then
  crunch usr.share.grub usr/share/grub &
else
  crunch usr.lib.grub usr/lib/grub &
```

```
fi

# Save the scripts we will use to restore.
cp -p /etc/bare.metal.recovery/* "${zip}"/bin &

# For sneaking our own ssh config onto the restore disk.
crunch root.ssh root/.ssh &
crunch etc.ssh etc/ssh &

wait
echo "Testing our results."
find "${zip}" -iname "*.bz2" -print0 | xargs -0 bunzip2 -t &

# Since we're doing system stuff anyway, make a boot disk ISO image
# suitable for burning. It uses the current kernel.

if [ "${REDHAT}" == 0 ] ; then
  mkbootdisk --iso --device "${zip}/bootdisk.$(uname -r).iso" "$(uname -r)"
fi

# 2007ish kernels incorporated a new ATA (IDE) hard drive
# driver. Because of this, parallel ATA (PATA) drives show up as SCSI
# drives, as serial ATA have always done. However, not all rescue
# distributions (e.g. finnix) use this new driver. So the following
# line very carefully replaces "/dev/sda" with "/dev/hda". Use this as
# a template if you have multiple IDE hard drives. Newer versions of
# finnix have PATA drives show up as SCSI drives (sd), so we have
# commented it out. It is commented out so you can comment it in if
# you need it.

# Note that there is no guaranteed mapping! Systems with multiple hard
# drives may have confusing mappings. Be sure to edit this line
# carefully. Check it if you add or remove a hard drive of any
# interface type to or from your system!

# find "${zip}" -type f | grep -v bz2$ | xargs sed -i 's|/dev/sda|/dev/hda|g'

pushd ${zips} || exit
echo "Making checksums so we can verify..."
find "${HOSTNAME}.${today}" -type f | sort | xargs sha512sum >  "${zip}".sha512sums &
find "${HOSTNAME}.${today}" -type f | sort | xargs sha256sum >  "${zip}".sha256sums &
popd || exit

du -hs "${zip}"
df -h

wait

echo Sucess!
```

11.1.10. `restore.metadata`

This script restores metadata from the ZIP disk as a first stage restore.

N.B: If you have libata IDE drive issues, the grub-install line at the end of this script won't work. If it doesn't, use your rescue disk to do the same.

N.B: Be sure to edit the list of directories to be restored to suit your system.

This script runs **grub-install** toward the end. It may fail. In that case, Boot-Repair (https://sourceforge.net/p/boot-repair/home/Home/) may repair it for you. Be sure to check options before letting it do its thing.

```
#! /bin/sh

# A script to restore the meta-data from the ZIP disk. This runs under
# tomsrtbt only after partitions have been rebuilt, file systems made,
# and mounted. It also assumes the ZIP disk has already been
# mounted. Mounting the ZIP disk read only is probably a good idea.

# Copyright 2000 through the last date of modification Charles Curley.

# This program is free software; you can redistribute it and/or modify it
# under the terms of the GNU General Public License as published by the
# Free Software Foundation; either version 2 of the License, or (at your
# option) any later version.

# This program is distributed in the hope that it will be useful, but
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
# General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
# 02110-1301, USA

# You can also contact the Free Software Foundation at http://www.fsf.org/

# 2007-07-08: We now force inittab to run level 3, which may be disty
# dependent. FHS compliance changes as well.

# 2005-08-03: We now use a relative path, so you can load from
# different places depending on the first stage system you are
# using. Also added some FC4 tricks, and some changes to better
# reproduce the permissions and ownerships.

# 2003 08 23: Oops: tar on tomsrtbt does not respect -p. Try setting
# umask to 0000 instead.

# 2003 02 13: Tar was not preserving permissions on restore. Fixed
# that.

# 2002 07 01: Went to bzip2 to compress the archives, for smaller
# results. This is important in a 100MB ZIP disk. Also some general
# code cleanup.
```

```
# For more information contact the author, Charles Curley, at
# http://www.charlescurley.com/.

umask 0000

cd ..                     # Assume we are in /bin
zip=$(pwd)/data;          # Where we find the tarballs to restore.
target="/target";         # Where the hard drive to restore is mounted.

ls -lt "$zip"/*.bz2         # Warm fuzzies for the user.

# shellcheck disable=SC3044
cd $target || { echo Cd to $target failed! ; exit ; }

# Restore the archived metadata files.
for archive in "$zip"/*.bz2; do
  echo "$archive"
  ls -al "$archive"
  bzip2 -dc "$archive" | tar -xf - &
done
wait

# Build the mount points for our second stage restoration and other
# things.

# If you boot via an initrd, make sure you build a directory here so
# the kernel can mount the initrd at boot. tmp/.font-unix is for the
# xfs font server.

# Be sure to edit the list of directories to be restored to suit your
# system(s)!
for dir in\
   boot/efi\
   dev\
   initrd\
   media/disk\
   media/installer\
   mnt\
   proc\
   sys\
   tmp/.font-unix\
   usr/share\
   var/lib/bare.metal.recovery\
   var/lock/subsys\
   var/log\
   var/run\
   var/spool\
 ; do

  mkdir -p $target/$dir
done
```

```
for dir in mnt usr usr/share $(ls -d var/*) selinux usr/lib var \
  var/cache/yum var/lock/subsys var/empty/sshd/etc \
  var/spool media ; do
  chmod go-w $target/"$dir"
done

# Set modes
# chmod 777 $target/var/lock
# chmod o+t $target/var/lock
chmod 777 $target/run/lock
chmod o+t $target/run/lock

ln -s run/lock var/lock
ln -s run var/run
# shellcheck disable=SC3044
cd - || { echo Cd - from $target failed! ; exit ; }

# For Fedora. First two for xfs.
# chroot $target chown xfs:xfs /tmp/.font-unix
# chmod 1777 $target/tmp/.font-unix # set the sticky bit.
chmod 1777 $target/tmp

# Set the system to boot to run level 3 regardless of the current run
# level. Be sure to set it back to the normal value.

if [ -e $target/etc/inittab ] ; then
  sed -i s/id:.:initdefault:/id:3:initdefault:/g $target/etc/inittab
fi

df -m
```

11.1.11. `first.stage`

This script runs the entire first stage restore with no operator intervention.

If you want to check for bad blocks when it puts a file system on the partitions, use a "-c" command line option.

```
#! /bin/bash

# A master script to run the other, detailed scripts. Use this script
# only if you want no human intervention in the restore process. The
# only option is -c, which forces bad block checking during formatting
# of the partitions.

# Copyright 2002 through the last date of modification Charles Curley.

# This program is free software; you can redistribute it and/or modify it
# under the terms of the GNU General Public License as published by the
# Free Software Foundation; either version 2 of the License, or (at your
# option) any later version.

# This program is distributed in the hope that it will be useful, but
```

```
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
# General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
# 02110-1301, USA

# You can also contact the Free Software Foundation at http://www.fsf.org/

# For more information contact the author, Charles Curley, at
# http://www.charlescurley.com/.

# 2005-08-07 We no longer assume the working directory. This is
# because the working directory will vary greatly according to which
# Linux disty you use and how you are doing your restoration.

export blockcheck=$1;

if [ "$blockcheck" != "-c" ] && [ -n "$blockcheck" ]
then
    echo "${0}: automated restore with no human interaction."
    echo "${0}: -c: block check during file system making."
    exit 1;
fi

for drive in $( ls make.dev.* ); do
    echo "$drive"$'\a'
    sleep 2
    ./"$drive" "$blockcheck";
done

# If there are any LVM volumes, now is the time to restore them.

if [ -e make.lvs ] && [ -e mount.lvs ]
then
    echo make.lvs$'\a'
    sleep 2
    ./make.lvs

    echo mount.lvs$'\a'
    ./mount.lvs
fi


# WARNING: If your Linux system mount partitions across hard drive
# boundaries, you will have multiple "mount.dev.* scripts. You must
# ensure that they run in the proper order, which the loop below may
# not do. The root partition should be mounted first, then the rest in
# the order they cascade. If they cross mount, you'll have to handle
# that manually. If you have LVMs to deal with, that's a whole 'nother
# kettle of fish.
```

```
# The "ls -tr" will list the scripts in the order they are created, so
# it might be a good idea to create them (in the script save.metadata)
# in the order in which you should run them.

for drive in $( ls -tr mount.dev.* ); do
    echo "$drive"$'\a'
    sleep 2
    ./"$drive";
done

./restore.metadata

echo Installing grub now.
./install.grub

# People who are really confident may comment this line in.
# reboot
```

11.2. Second Stage

These scripts run on the computer being backed up or restored.

11.2.1. `back.up.all`

This script saves to another computer via an NFS mount. You can adapt it to save to tape drives or other media.

```
#! /bin/bash

# Back up the entire system to another computer's drive. To make this
# work, we need a convenient chunk of disk space on the remote computer we
# can nfs mount as /mnt/save.

# Copyright 2000 through the last date of modification Charles Curley.

# This program is free software; you can redistribute it and/or modify it
# under the terms of the GNU General Public License as published by the
# Free Software Foundation; either version 2 of the License, or (at your
# option) any later version.

# This program is distributed in the hope that it will be useful, but
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
# General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
# 02110-1301, USA
```

```
# You can also contact the Free Software Foundation at http://www.fsf.org/

# For more information contact the author, Charles Curley, at
# http://www.charlescurley.com/.

save="/mnt/save"

# Make sure it's there
umount $save
mount $save

cd / || exit

rm $save/tester.tar.old.gz
mv $save/tester.tar.gz $save/tester.tar.old.gz

# save everything except /mnt, /proc, and nfs mounted directories.

time tar cf - / --exclude /mnt --exclude /proc --exclude $save\
    | gzip -c > $save/tester.tar.gz
```

11.2.2. `back.up.all.ssh`

This script does exactly what `back.up.all` does, but it uses SSH instead of NFS.

```
#! /bin/bash

# Back up the entire system to another computer's drive. To make this
# work, we need a convenient chunk of disk space on the remote
# computer. This version uses ssh to do its transfer, and compresses
# using bz2. This means this script has to know more about the other
# computer, which does not make for good modularization.

# Copyright 2000 through the last date of modification Charles Curley.

# This program is free software; you can redistribute it and/or modify it
# under the terms of the GNU General Public License as published by the
# Free Software Foundation; either version 2 of the License, or (at your
# option) any later version.

# This program is distributed in the hope that it will be useful, but
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
# General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
# 02110-1301, USA

# You can also contact the Free Software Foundation at http://www.fsf.org/
```

```
# For more information contact the author, Charles Curley, at
# http://www.charlescurley.com/.

save="/backs/tester"
backup_server="charlesc"

# rotate the old backups. Do it all in one line to minimze authentication overhead.
ssh $backup_server "rm $save/tester.tar.old.bz2; mv $save/tester.tar.bz2 \
    $save/tester.tar.old.bz2"

# save everything except /mnt, /proc, and squid directories.

time tar cf - / --exclude /mnt --exclude /proc --exclude /var/spool/squid\
    | ssh $backup_server "bzip2 -9 > $save/tester.tar.bz2"
```

11.2.3. `restore.all`

This is the restore script to use if you backed up using `back.up.all`.

```
#! /bin/bash

# A script to restore all of the data from an nfs mount. This is our final
# stage restore.

# Copyright 2000 through the last date of modification Charles Curley.

# This program is free software; you can redistribute it and/or modify it
# under the terms of the GNU General Public License as published by the
# Free Software Foundation; either version 2 of the License, or (at your
# option) any later version.

# This program is distributed in the hope that it will be useful, but
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
# General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
# 02110-1301, USA

# You can also contact the Free Software Foundation at http://www.fsf.org/

# For more information contact the author, Charles Curley, at
# http://www.charlescurley.com/.

export save="/mnt/save"

mount $save

cd /
gunzip -dc $save/tester.tar.gz | tar -xpkf -
```

```
rm /var/run/*.pid

lilo
```

11.2.4. `restore.all.ssh`

This is the restoration script to use if you used `back.up.all.ssh` to back up.

```
#! /bin/bash

# A script to restore all of the data using ssh and bunzip2. This is
# our final stage restore.

# Copyright 2000 through the last date of modification Charles Curley.

# This program is free software; you can redistribute it and/or modify it
# under the terms of the GNU General Public License as published by the
# Free Software Foundation; either version 2 of the License, or (at your
# option) any later version.

# This program is distributed in the hope that it will be useful, but
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
# General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
# 02110-1301, USA

# You can also contact the Free Software Foundation at http://www.fsf.org/

# For more information contact the author, Charles Curley, at
# http://www.charlescurley.com/.

save="/backs/tester/"
backup_server="charlesc"

cd /

ssh $backup_server "cat $save/tester.tar.bz2" | bunzip2 | tar -xpkf -

rm /var/run/*.pid

lilo
```

11.3. Backup Server Scripts

The SSH scripts above have a possible security problem. If you run them on a firewall, the firewall has to have access via SSH to the backup server. In that case, a clever cracker might also be able to crack the backup server. It would be more secure to run backup and restore scripts on the backup server, and let the backup server have access to the firewall. That is what these scripts are for.

These scripts back up and restore the target completely, not just the stage one backup and restore. `get` backs up the bare metal archive separately so that you can make a CD-ROM or NFS mount from it.

I use these scripts routinely.

11.3.1. `get`

```
#! /bin/bash

# Back up another computer's drive to this system. To make this work,
# we need a convenient chunk of disk space on this computer. This
# version uses ssh to do its transfer, and compresses using bz2. This
# version was developed so that the system to be backed up won't be
# authenticated to log onto the backup computer. This script is
# intended to be used on a firewall. You don't want the firewall to be
# authenticated to the backup system in case the firewall is cracked.

# Copyright 2000 through the last date of modification Charles Curley.

# This program is free software; you can redistribute it and/or modify it
# under the terms of the GNU General Public License as published by the
# Free Software Foundation; either version 2 of the License, or (at your
# option) any later version.

# This program is distributed in the hope that it will be useful, but
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
# General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
# 02110-1301, USA

# You can also contact the Free Software Foundation at http://www.fsf.org/

# For more information contact the author, Charles Curley, at
# http://www.charlescurley.com/.

# 2007-05-22: Changes for FHS compliance. Removed commented out
# references to ZIP drives. Now, if the archive does not exist on the
# client, we run save.metadata there.

# 2006-04-14: Use the script ../scripts/get.target to determine the
# host name. This makes the script more general, and maybe we can run
# it from one place & determine the host name from the directory
```

```
# name. Also, we now test to see if old backups exist before deleting
# them.

# 2004 04 03: added /sys to the list of excludes. It is a read-only
# pseudo-file system like /proc.

# 2002 07 01: We now set the path on the target to the zip drive with
# a variable. This fixes a bug in the command to eject the zip disk.

# 2002 07 01: The zip disk archives are now in bzip2 format, so this
# script has been changed to reflect that.

# Get the host name of the computer to be backed up and other info.
. ../scripts/get.target

# The "--anchored" option is there to prevent --exclude from excluding
# all files with that name. E.g. we only want to exclude /sys, not
# some other sys elsewhere in the file system.

ssh $host "cd / ; tar -cf - --anchored --exclude media --exclude mnt\
 --exclude selinux --exclude sys --exclude proc --exclude var/spool/squid\
 --exclude var/cache/yum --exclude var/named/chroot/proc\
 --exclude var/lib/bare.metal.recovery * " | bzip2 -9 | cat > $host.$DATE.tar.bz2

# if [ -e $host.dos.$DATE.old.tar.bz2 ] ; then
#     rm $host.dos.$DATE.old.tar.bz2
# fi

# echo Backing up $host dos to the backup server.
# ssh $host "cd / ; mount mnt/dosc ; tar -cf - mnt/dosc "\
# | bzip2 -9 | cat > $host.dos.$DATE.tar.bz2

echo Testing the results.
find $host.$DATE* -iname "*.bz2" | xargs bunzip2 -t

# Post-processing.
. ../scripts/post.sh
```

11.3.2. `post.sh`

This is called at the end of scripts to get target backups, such as `get`. It does some post-processing to be sure we have transferred the data from the target to the server correctly.

Note that it expects and tests copies of the checksums made on the target. It also creates new checksum files, which include the archive tarball which `get` creates.

```
# -*- shell-script -*-

# Copyright 2018 through the last date of modification, Charles Curley.

# This program is free software; you can redistribute it and/or modify it
# under the terms of the GNU General Public License as published by the
```

```
# Free Software Foundation; either version 2 of the License, or (at your
# option) any later version.

# This program is distributed in the hope that it will be useful, but
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
# General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
# 02110-1301, USA

# You can also contact the Free Software Foundation at
# http://www.fsf.org/

# Do post-processing, such as caclulating check sums, for a recently
# completed archive. Call from the archive directory's get script,
# after it has run get.archive and especially get.target.

echo
echo Doing sha512 checksums...
find $host.$DATE $host.$DATE.*.bz2 -type f | sort | xargs sha512sum >  $host.$DATE.all.sha512sums
find $host.$DATE $host.$DATE.*.bz2 -type f | sort | xargs sha256sum >  $host.$DATE.all.sha256sums

# Test copied files
sha512sum --quiet -c $host.$DATE.sha512sums
sha256sum --quiet -c $host.$DATE.sha256sums
```

11.3.3. `get.target`

```
# -*- shell-script -*-

# Copyright 2000 through the last date of modification, Charles Curley.

# This program is free software; you can redistribute it and/or modify it
# under the terms of the GNU General Public License as published by the
# Free Software Foundation; either version 2 of the License, or (at your
# option) any later version.

# This program is distributed in the hope that it will be useful, but
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
# General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
# 02110-1301, USA

# You can also contact the Free Software Foundation at
```

```
# http://www.fsf.org/

# How to determine the target: the directory structure is
# /..../back/hostname.OS.etc So we have to get the name of the
# directory. That gives us host name and OS.etc. From that we extract
# the host name.

path=$(pwd)

target=${path##/*/}
host=${target%%.*}

# Run "info date" for more information.

DATE=$(date +%Y%m%d)
# echo "Today's date is $DATE."

echo "\$target is $target. \$host is $host. Today's date is $DATE."

name=$0
echo "This is script $name"

if [ $(echo "$name" | grep -i get > /dev/null) ] ; then
    # Do functions common to all restores.

    # Which archive do we restore and is this a valid target name?

    TARGET=$1

    if [ -z "$TARGET" ] ; then
        echo Please specify a target from one of:
        for dir in $(ls -d "$host".*) ; do
            if [ -d "$dir" ] ; then
                echo -n "$dir "
            fi
        done
        exit 2;
    fi

    if [ -z "$TARGET" ] || [ ! -d "$TARGET" ] ; then
        echo "$TARGET" does not exist!
        exit 2;
    fi

    ssh "$host" rm -r /var/lib/rpm
else
    # Do functions common to all gets.

    # Where we will get the archives on the target.

    #zip=/mnt/zip
    export zip="/var/lib/bare.metal.recovery";
```

```
    # If it does not already exist, build the archive.

    ssh "$host" "if [ ! -d ${zip}/$DATE ] ; then echo Saving metadata... ; save.metadata ; fi"

    echo Backing up "$host"

    if [ -e "$host.$DATE" ] ; then
     rm -r "$host.$DATE"
    fi

    echo Copying the bare metal recovery archive.

    # -r for recursive copy, -p to preserve times and permissions, -q
    # for quiet: no progress meter.

    scp -qpr "$host:$zip/$DATE" "$host.$DATE"

    du -hs "$host".*

    echo Cleaning out old yum packages
    ssh "$host" "yum clean packages"

    echo Backing up "$host" to the backup server.

    if [ -e "$host.$DATE".tar.bz2 ] ; then
        rm "$host.$DATE".tar.bz2
    fi
fi
```

11.3.4. restore

```
#! /bin/bash

# A script to restore all of the data to tester via ssh. This is our final
# stage restore.

# Copyright 2000 Charles Curley.

# This program is free software; you can redistribute it and/or modify it
# under the terms of the GNU General Public License as published by the
# Free Software Foundation; either version 2 of the License, or (at your
# option) any later version.

# This program is distributed in the hope that it will be useful, but
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
# General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
```

```
# 02110-1301, USA

# You can also contact the Free Software Foundation at http://www.fsf.org/

# For more information contact the author, Charles Curley, at
# http://www.charlescurley.com/.

# 2007-05-22: Changes for FHS compliance. Removed commented out
# references to ZIP drives.

# 2006-04-14: Use the script ../scripts/get.target to determine the
# host name. This makes the script more general, and maybe we can run
# it from one place & determine the host name from the directory
# name.

# Get the host name of the computer to be backed up and other info.
. ../scripts/get.target

bunzip2 -dc $TARGET.tar.bz2 | ssh $host "umask 000 ; cd / ; tar -xpkf - "

# bunzip2 -dc $host.dos.$DATE.tar.bz2 | ssh $host "umask 000 ;\
# mount /mnt/dosc ; cd / ; tar -xpkf - "

# Note libata issue! We boot to /dev/sda, not /dev/hda, as IDE drives
# now show up as SCSI drives.

ssh $host "chown root:lock /var/lock ; grub-install /dev/sda"
# ; chown -R amanda:disk /var/lib/amanda
```

11.4. Miscellaneous Files

11.4.1. `install`

This little script just installs things on target systems and sets up a few directories. It tests for the presence of the perl UUID module.

It would be a useful basis for an RPM or deb package. The placement of files is based on the *Filesystem Hierarchy Standard (http://www.pathname.com/fhs/)*, version 2.3, announced on January 29, 2004.

```
#! /bin/bash

# A script to install the bare metal recovery scripts. With any luck,
# this will comply with the "Filesystem Hierarchy Standard",
# http://www.pathname.com/fhs/, version 2.3, announced on January 29,
# 2004.

# Copyright 2007 through the last date of modification, Charles Curley.

# This program is free software; you can redistribute it and/or modify it
# under the terms of the GNU General Public License as published by the
```

```
# Free Software Foundation; either version 2 of the License, or (at your
# option) any later version.

# This program is distributed in the hope that it will be useful, but
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
# General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
# 02110-1301, USA

# You can also contact the Free Software Foundation at
# http://www.fsf.org/

# 2017-07-12: We now check to see if the perl UUID module is installed
# and in perl's module path.

# 2014-01-06: we now install a set of excludes for use in general
# backing up of stuff not backed up by the general backup system
# (amanda, e.g., or by the metatdata/minimal system backups.

# Begin the actual code.

# Check for the presence of perl's UUID module (and, indirectly, perl
# itself).

perl -e 'use UUID;'
uuid=$?

if [ "$uuid" != "0" ] ; then
    echo Oops! Please install perl module UUID.
    echo E.g.: apt install libuuid-perl
    echo Perl exited with a return value of $uuid.
    exit $uuid
fi

# Where we put the archives ready for making CDs, etc.
mkdir -p /var/lib/bare.metal.recovery

# Keep them secure from pesky snooping users who don't need to be able
# to hack a copy of, say, /etc/shadow.
chmod 700 /var/lib/bare.metal.recovery

# Excludes for use by the "get" script on the bare metal backup
# server. Make them read/writeable only by root.
cp ../bare.metal.backup.excludes /var/lib/bare.metal.recovery
chmod 0600 /var/lib/bare.metal.recovery/bare.metal.backup.excludes

# Backup time executables.
executables="save.metadata make.fdisk"
gpts=$(for i in /dev/[sv]d? /dev/nvme*n? ; do fdisk -l "$i" | grep -i gpt ; done )
```

```
if [ -n "${gpts}" ] ; then
    executables="$executables make.gdisk"

    if test -z "$(command -v fsarchiver)" ; then
        if [ -d /boot/efi ] ; then
            printf "Please install fsarchiver for EFI systems!\n"
            printf "E.g.: \"apt -y install fsarchiver\".\n"
            exit 2
        fi
    fi
fi

for i in $executables ; do
  cp -rp "$i" /usr/sbin
  chown root:root /usr/sbin/"$i"
done

# Save the recovery time executables we provide. The archiving
# programs look for them here and save them into the archives.
mkdir -p /etc/bare.metal.recovery
for i in first.stage restore.metadata install.grub ssh.sh ; do
  cp -rp $i /etc/bare.metal.recovery
  chown root:root /etc/bare.metal.recovery/$i
done

if [ ! "$(grep -c '^processor[[:blank:]]*: [[:digit:]]*$' /proc/cpuinfo)" = "1" ] &&
       test -z "$(command -v pbzip2)" ; then
    echo "Multiprocessor! This program would benefit from pbzip2.";
    echo "E.g.: apt install pigz pbzip2";
fi
```

11.4.2. `ssh.sh`

A handy script to run on the restoration system so you can log in as usual to the system. After running this, you might log out on the target's terminal so that the history will be available when you log in over ssh.

```
#! /bin/sh

# A script to install the ssh configuration from the backups.

# If the sshd service fails to run, try starting it rather than
# reloading it.

# systemctl start sshd.service
# systemctl status sshd.service

# Copyright 2026 Charles Curley.

# This program is free software; you can redistribute it and/or modify it
# under the terms of the GNU General Public License as published by the
```

```
# Free Software Foundation; either version 2 of the License, or (at your
# option) any later version.

# This program is distributed in the hope that it will be useful, but
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
# General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
# 02110-1301, USA

# You can also contact the Free Software Foundation at
# http://www.fsf.org/

mtpt=$(pwd)

cd /etc || { printf "cd to /etc failed.\n"; }

if ! mv ssh ssh.old ; then
    printf "Failed to move ssh aside.\n"
    exit 1
fi

cd / || { printf "cd to / failed.\n"; }
tar xvf "$mtpt/../data/etc.ssh.tar.bz2"


cd /root || { printf "cd to /root failed.\n"; }

if [ -d .ssh ] ; then
    if ! mv .ssh .ssh.old ; then
        printf "Failed to move .ssh aside.\n"
        exit 1
    fi
    cd / || { printf "cd to / failed.\n"; }
    tar xvf "$mtpt/../data/root.ssh.tar.bz2"
fi

systemctl reload-or-restart sshd.service
systemctl status sshd.service
```

12. Resources

In no particular order. These are things you might want to investigate for yourself. A listing here should not be taken as an endorsement. In fact, in many case I have not used the product and cannot comment on it.

- Network-booting Your Operating System (http://osdev.berlios.de/netboot.html) describes several techniques for booting across a network, using grub (http://www.gnu.org/software/grub/) and some other tricks. I haven't tried it, but I have a sneaky suspicion that with an especially trained floppy diskette, you could get your entire first stage image onto the computer to be restored.
- "Smart Boot Manager (SBM) (http://btmgr.webframe.org/) is an OS independent and full-featured boot manager with an easy-to-use user interface. There are some screen shots available." It is essential if your BIOS will not allow you to boot to CD-ROM and you want to use a CD-ROM based Linux for Stage 1 recovery.
- W. Curtis Preston (http://www.oreilly.com/catalog/unixbr/author.html)'s excellent *Unix Backup & Recovery* (http://www.oreilly.com/catalog/unixbr/). This is the book that got me started on this bare metal recovery stuff. I highly recommend it; read my review (http://www2.linuxjournal.com/lj-issues/issue78/3839.html). However, you should probably get the latest edition.
- tomsrtbt (http://www.toms.net/rb/), "The most Linux on 1 floppy disk." Tom also has links to other small disties.
- The Linux Documentation Project (http://www.tldp.org/). See particularly the "*LILO, Linux Crash Rescue HOWTO*."
- The Free Software Foundation's `parted` (http://www.gnu.org/software/parted) for editing (enlarging, shrinking, moving) partitions.
- QtParted (http://qtparted.sourceforge.net/) looks to do the same thing with a GUI front end.
- Partition Image (http://www.partimage.org/Main_Page) for backing up partitions.

  From the web page: "Partition Image is a Linux/UNIX utility which saves partitions in many formats (see below) to an image file. The image file can be compressed in the GZIP/BZIP2 formats to save disk space, and split into multiple files to be copied on removable floppies (ZIP for example), ... The partition can be saved across the network since version 0.6.0."
- Bacula (http://sourceforge.net/projects/bacula) is a GLPled backup product which has bare metal recovery code inspired in part by this HOWTO.
- "g4u ('ghost for unix') (http://www.feyrer.de/g4u/) is a NetBSD-based bootfloppy/CD-ROM that allows easy cloning of PC harddisks to deploy a common setup on a number of PCs using FTP. The floppy/CD offers two functions. First is to upload the compressed image of a local harddisk to a FTP server. Other is to restore that image via FTP, uncompress it and write it back to disk; network configuration is fetched via DHCP. As the harddisk is processed as a image, any filesystem and operating system can be deployed using g4u."
- "We present Frisbee (http://www.cs.utah.edu/flux/papers/frisbee-usenix03-base.html), a system for saving, transferring, and installing entire disk images, whose goals are speed and scalability in a LAN environment. Among the techniques Frisbee uses are an appropriately-adapted method of filesystem-aware compression, a custom application-level reliable multicast protocol, and flexible application-level framing. This design results in a system which can rapidly and reliably distribute a disk image to many clients simultaneously. For example, Frisbee can write a total of 50 gigabytes of data to 80 disks in 34 seconds on commodity PC hardware. We describe Frisbee's design and implementation, review important design decisions, and evaluate its performance."
- There are a number of USB key disties available. Check DistroWatch (http://distrowatch.com/) for details.
- CD-ROM based rescue kits. This is not intended to be an exhaustive list. If you know of one (or even something that pretends to be one), please let me know (charlescurley at charlescurley dot com). You may find more recent information at DistroWatch (http://www.distrowatch.com/).
  - Boot-Repair (https://sourceforge.net/p/boot-repair/home/Home/) will repair your Grub 2 boot system. It is useful if the call to **grub-install** in `restore.metadata` fails.

- Mondo Rescue (http://freshmeat.net/projects/mondorescue) "... creates one or more bootable Rescue CD's (or tape+floppies) containing some or all of your filesystem. In the event of catastrophic data loss, you will be able to restore from bare metal."
- The Crash Recovery Kit for Linux (http://crashrecovery.org/)
- "System recovery with Knoppix" (http://www-106.ibm.com/developerworks/linux/library/l-knopx.html?ca=dgr-lnxw04Knoppix) is a good introduction to system recovery in general, and has some useful Knoppix (http://www.knoppix.org/) links.
- "Cool Linux CD (http://emergencycd2.sourceforge.net/) is live CD with Linux system. This used 2.4 kernel and some free and demo soft."
- SystemRescueCd (http://www.sysresccd.org/Index.en.php)" is a linux system on a bootable cdrom for repairing your system and your data after a crash. It also aims to provide an easy way to carry out admin tasks on your computer, such as creating and editing the partitions of the hard disk. It contains a lot of system utilities (parted, partimage, fstools, ...) and basic ones (editors, midnight commander, network tools). It aims to be very easy to use: just boot from the cdrom, and you can do everything. The kernel of the system supports most important file systems (ext2/ext3, reiserfs, xfs, jfs, vfat, ntfs, iso9660), and network ones (samba and NFS)."
- Syslinux (http://syslinux.zytor.com/) builds boot code for floppy diskettes, CD-ROMs and Intel PXE (Pre-Execution Environment) images. It is not dependent on a floppy diskette image. You can build your own CDs with a number of tools, such as tomsrtbt (http://www.toms.net/rb/), on it.
- In case you'd like to roll your own: "Linux Live (http://www.linux-live.org/) is a set of bash scripts which allows you to create [your] own LiveCD from every Linux distribution. Just install your favourite distro, remove all unnecessary files (for example man pages and all other files which are not important for you) and then download and run these scripts."
- "The PPART CD (http://www.linbox.com/en/ppart.html) allows you to generate system recovery bootable CD of previously saved hard disks."
- Timo's Rescue CD Set (http://rescuecd.sourceforge.net/): " This set is my approach for an easy way to generate a rescue system on a bootable cd, which can easily be adapted to the own needs. The project evolves more and more into a 'debian on cd' project, so it's not only possible to use the system as a rescuecd, it is also possible to install a whole debian system on cd."
- The List of Live CDs (http://www.livecdlist.com/) has more CD based disties.

A. GNU Free Documentation License

Version 1.1, March 2000

Copyright (C) 2000 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially

or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any

title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

11. How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Notes

1. I emphasize copy because **mkisofs** will mung the file in the directory from which it makes the ISO image.