Linux Complete Backup and Recovery HOWTO

Charles Curley

<charlescurley at charlescurley dot com>

Revision History

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 Amanda.		
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 runs the entire first stage.		
Revision 0.05	2002−07−04	Revised by: c^2
Added another variation, other minor fixes.		
Revision 0.04	2002−05−09	Revised by: c^2
Added Theme & Variations section, a variation; also some minor reorganization.		
Revision 0.03	2002−04−09	Revised by: c^2
Added the Beeblebrox Project to resources.		
Revision 0.02	2002−01−27	Revised by: c^2
Added Mondo to resources		
Revision 0.01	2001−10−25	Revised by: c^2
Initial version for LDP release		

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 Dal) 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.

Table of Contents

Table of Contents

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

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 or anyone else. Use the concepts, examples and other content at your own risk. There may be errors and inaccuracies that may be damaging to your system. Proceed with caution, and although errors are unlikely, the author(s) 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, espcially the scripts, in this HOWTO.

1.3. New Versions

You can find this document at its home page or at the Linux Documentation Project homepage 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.

- bzip2 compressed chunky (lots of small pages. Faster reading.) HTML.
- bzip2 compressed smooth (one monster page −− no chunks. Easier to search.) HTML.

- bzip2 compressed postscript (US letter format).
- bzip2 compressed PDF (US letter format).
- bzip2 compressed raw ASCII text.
- Use the source, Luke.

1.4. Credits

This document is derived from two articles originally published in *Linux Journal*. 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.

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*!

The target computer for this HOWTO is a Pentium computer with a Red Hat 7.1 Linux server or workstation installation on one IDE hard drive. 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*, O'Reilly & Associates, 1999, which I have favorably reviewed in *Linux Journal*. However, the book is a bit thin on specific, real−time questions. For example, exactly which files do you back up? What metadata do you need to preserve, and how?

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 Red Hat Package Manager (RPM) based Linux distributions should also save RPM metadata as part of their normal backups. Something like:

```
bash# rpm −Va > /etc/rpmVa.txt
```

in your backup script will give you a basis for comparison after a bare metal restoration.

To get to this point, you need to 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.
- A parallel port Iomega® ZIP® drive or equivalent. You will need at least 30 MB of space.
- Your backup media.
- A minimal Linux system that will allow you to run the restoration software.

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 minimal file system 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 huge 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 and 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 ext2fs, 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. Partition Image 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 rescue disk. I use tomsrtbt. It is well documented and packs a lot of useful tools onto one floppy diskette. There is an active list for it, and the few questions I've had were quickly and accurately answered. I like that in a product my shop may depend on one day.

Next, figure out how to do the operating system backup you will need so that you can restore your normal backup. I followed Preston's advice and used 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.

3.1. Installing the ZIP Drive

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

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 script `make.fdisk` scans a hard drive for partition information, and saves it in three files. The first is an executable script, called `make.dev.x` (where "x" is the name of the device file, e.g. hda). Second is `mount.dev.x`, which creates mount points and mounts the newly created partitions on them. The last, `dev.x`, is the commands necessary for **fdisk** to build the partitions. 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 file for **fdisk**, `dev.hda`.

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`

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. Right now, that's the only way to rebuild on a larger replacement drive. (See the To Do list.)

Other metadata are preserved in the script `save.metadata`. 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.

To determine the directories needed at boot, we look at the boot initialization file `/etc/rc.sysinit`. 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, such as /dev. In Linux, you can't do much without device files.

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
```

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

4.1. Theme And Variations

4.1.1. No ZIP drive

This backup process requires you to have the ZIP disk drive present at each backup. You could, given a large enough hard drive, create the ZIP disk's contents in a directory, and back that 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, but you should check that the resulting directory will fit on your ZIP disk (with **du −hs $target.zip**)! You will have to edit the definition of the variable `zip` in `save.metadata`.

My laptop has problems running both a network card and a ZIP drive, so this is the process I use to back it up.

Alternatively, you could build several ZIP disks' worth of backup on the hard drive, and feed them to the system on restore.

4.1.2. 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.

4.1.3. Excluding From First Stage Saving

There are time when you need to squeeze a few megabytes from the first stage data, expecially 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 **emacs** and **gs** like so:

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

Why those two? Because they're hard drive space hogs and we don't need them at boot time.

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

4.1.4. Initrd

If your system uses an initial RAM disk, or initrd, to boot, make sure that `save.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` to see if it calls for one.

5. First Stage Restore

5.1. Booting tomsrtbt

The first thing to do before starting the restoration process 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.

Before booting tomsrtbt, make sure your ZIP drive is placed 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

Once tomsrtbt has booted and you have a console, mount the ZIP drive. It is probably a good idea to mount it read only:

```
# mount /dev/sda1 /mnt −o ro
```

Check to be sure it is there:

```
# ls −l /mnt
```

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

To run the first stage restore completely automated, enter the command:

```
# /mnt/root.bin/first.stage
```

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

5.2.2. Manually

To run the process manually, change to the directory where the scripts are on the ZIP drive.

```
# cd /mnt/root.bin
```

Now run the script(s) that will restore the partition information and create file systems. You may run them in any 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 partitions from the information gathered when you ran `make.fdisk`.
- Make ext2 and ext3 file system 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.

NOTE

If you have other operating systems to restore, now is a good time to do so. When you've done that, reboot to tomsrtbt and continue restoring Linux.

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`. This will restore the contents of the ZIP drive to the hard drive.

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 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. To be safe, change your boot run level temporarily. In `/target/etc/inittab`, find the line that looks like this:

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

and change it to this:

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

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

```
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 on an RPM based system is to verify all packages:

```
bash# rpm −Va
```

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. You can redirect the output to a file, and diff it against the one that was made at backup time (/etc/rpmVa.txt), thereby speeding up this step considerably. Emacs users should check out its diff 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, and disabled it above, test X before you re−enable it. Re−enable it by changing that one line in /etc/inittab back to:

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

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. Red Hat 8.0

Red Hat 8.0 requires changes only if you log in via ssh to your computer after the first stage restore. If, say, you perform the second state restore over ssh. If so, make the following changes:

- In the file `save.metadata`, add the kerberos libraries to the first stage backup.
  ```
  crunch  kerberos usr/kerberos/lib/
  ```
- In the file `restore.metadata`, there is a loop that builds several directories. First, add these two directory names to the list: `/var/empty/sshd` and `/var/lock/subsys`. As Red Hat 8.0 defaults to ext3fs, it requires an inital RAM disk at boot time. So make sure `initrd` is in the list. Then, if it isn't already there, add the **`-p`** switch to the makdir command.

  The directory `/var/lock/subsys` is owned by group lock, so change its owner.

  ```
  chroot $target /bin/chown root:lock /var/lock
  ```

  Finally, `usr/lib/libcrypto*` has gone away, so you can remove that from the line that crunches `usr/lib`.

7.2. Red Hat 7.1

This distribution is the one I used originally on my test computer. I have had no problems with it.

7.3. Red Hat 7.0

This version seems to require libcrack (in /usr/lib) and its attendant files in order to authenticate users. So in `save.metadata`, add to the line that saves /usr/lib the following: /usr/lib/*crack* and enable that line.

8. Application Specific Notes

I have listed below notes about backing up particular applications.

8.1. 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.tester`:

```
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

8.2. Arkeia

These notes are based on testing with Arkeia 4.2.

Arkeia 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 anlogous 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 dicy, 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 solve it on my own computer by saving the data dictionary to tape with The TOLIS Group's BRU.

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

8.3. Amanda

Amanda (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*. For more information on it, see the Resources. The Amanda chapter from the book is on line.

I made two changes to the script `restore.tester`. 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.

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 tomsrtbt floppies and possibly some ZIP drives in your off−site storage as well. Also, have copies of the tomsrtbt 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 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.

- A partition editor to adjust partition boundaries 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` 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` to recognize others and make appropriate instructions to rebuild them in the output files.
- For FAT12 or FAT16 partitions we do not format, write zeros into the partition so that Mess−DOS 6.x does not get confused. See the notes on **fdisk** for an explanation of the problem.
- Make a script for putting ext2/3 file systems on ZIP disks.
- Translations into other (human) languages.
- Find out how loadlin or similar programs affect this process.
- Changes for GRUB
- Change the scripts to use a CD−ROM. A CD−ROM that would boot to tomsrtbt, with the first stage restore data on the rest of it, would be just the ticket.
- I've referred to Red Hat Package Manager (rpm) from time to time. What are the equivalent deb commands?

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`, below. The name of the script and data file produced depends on the device given this script as a a parameter. That script, run at restore time, builds the partitions on the hard drive. `make.fdisk` is called from `save.metadata`, below.

```
#! /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.

# Time-stamp: <2003-04-24 09:58:15 ccurley make.fdisk>

# 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.,
# 59 Temple Place, Suite 330, Boston, MA 02111-1307 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:

# 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
# 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/.


# 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 {

print OUTPUT <<FINIS;

# 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.,
# 59 Temple Place, Suite 330, Boston, MA 02111−1307 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

}


# Begin main line code.



# Provide a default device.

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

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

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

# prepare format string.
$fmt = cut2fmt (11, 19, 24, 34, 45, 49);
# print "Format string is $fmt.\n\n";

# define fields in the array @_.
$dev = 0;
$bootable = 1;
$firstcyl = 2;
$lastcyl = 3;
$parttype = 5;
$partstring = 6;

$target = "\/target";

$outputfilename = $device;
$outputfilename =~ s/\//./g;
```

```
$outputfilename = substr ($outputfilename, 1, 100);

$outputfilepath = "\/root\/bin\/";


# 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?
#       print $_;               # print it just for grins
        split;
        if ($_[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";
        }


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

        if ($_[4] =~ /ext[23]/ and $_[5] eq "(rw)" ) {
            $mountpoints{$_[2]} = $_[0];
#           print "$_[2] is the mountpoint for $mountpoints{$_[2]}.\n";
        }
    }
}
close (MOUNT);


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

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

while (<FDISK>) {
    if ($_ =~ /^$device/i) {    # is this a line with a partition in it?
#       print $_;               # print it just for grins
        chop;                   # kill trailing \r
        @_ = unpack ($fmt, $_);

        # now strip white spaces from cylinder numbers
        @_[$firstcyl] =~ s/[ \t]+//;
        @_[$lastcyl] =~ s/[ \t]+//;
        @_[$parttype] =~ s/[ \t]+//;

        $partnumber = substr(@_[$dev], 8, 10); # get partition number for this line
        # just for grins
#       print "  $partnumber, @_[$firstcyl], @_[$lastcyl], @_[$parttype], @_[$partstring]\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";
                if (@_[$firstcyl] ne @_[$lastcyl]) { # in case it's all on one cylinder
                    print OUTPUT "@_[$lastcyl]\n";
                }

                # Now detect if this is an ext3 (journaling)
                # partition. We do this using dumpe2fs to dump the
                # partition and grepping on "journal". If the
                # partition is ext2, there will be no output. If it is
                # ext3, there will be output, and we use that fact to
                # set a command line switch. The command line switch
                # goes into an associative array (hash) so we don't
                # have to remember to reset it to the null string when
                # we're done.

                $dpid = open (DUMPE2FS, "dumpe2fs @_[$dev] 2>/dev/null | grep −i journal |")
                    or die "Couldn't fork: $!\n";
                while (<DUMPE2FS>) {
#                   print "Dumpe2fs: $_";
                    $ext3{$_[$dev]} = "−j ";
                    last;
                }
                close (DUMPE2FS);

                if ($labels{@_[$dev]}) { # do we have a label?
                    $format .= "echo\necho formatting $checking@_[$dev]\n";
                    $format .= "mke2fs $ext3{$_[$dev]}\$blockcheck −L $labels{@_[$dev]} @_[$dev]\
                } else {
                    $format .= "echo\necho formatting $checking@_[$dev]\n";
                    $format .= "mke2fs $ext3{$_[$dev]}\$blockcheck @_[$dev]\n\n";
                }

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

                # primary Linux swap partition
            } elsif (@_[$parttype] == 82) {
                print OUTPUT "p\n$partnumber\n@_[$firstcyl]\n";
                if (@_[$firstcyl] ne @_[$lastcyl]) {
                    print OUTPUT "@_[$lastcyl]\n";
                }
                print OUTPUT "t\n$partnumber\n82\n";
                $format .= "echo Making @_[$dev] a swap partition.\n";
                $format .= "mkswap \$blockcheck @_[$dev]\n\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" ) {
                # print ("Making DOS primary partition.\n");
                print ("dd if=@_[$dev] of=$outputfilepath$outputfilename$partnumber");
                print (" bs=512 count=1\n");
                system ("dd if=@_[$dev] of=$outputfilepath$outputfilename$partnumber bs=512 count
                print OUTPUT "p\n$partnumber\n@_[$firstcyl]\n";
                if (@_[$firstcyl] ne @_[$lastcyl]) { # in case it's all on one cylinder
                    print OUTPUT "@_[$lastcyl]\n";
                }
                print OUTPUT "t\n$partnumber\n@_[$parttype]\n";
                $format .= "echo\necho formatting $checking@_[$dev]\n";
                $format .= "mkdosfs \$blockcheck";
                if ( @_[$parttype] == b || @_[$parttype] == c) {
                    # 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$partnumber of=@_[$dev] bs=512 count=1\n\n";

            } else {
                # anything else partition
                print OUTPUT "p\n@_[$firstcyl]\n";
                if (@_[$firstcyl] ne @_[$lastcyl]) {
                    print OUTPUT "@_[$lastcyl]\n";
                }
                print OUTPUT "t\n$partnumber\n@_[$parttype]\n";
            }

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

                # Now detect if this is an ext3 (journaling)
                # partition. We do this using dumpe2fs to dump the
                # partition and grepping on "journal". If the
                # partition is ext2, there will be no output. If it is
                # ext3, there will be output, and we use that fact to
                # set a command line switch. The command line switch
                # goes into an associative array (hash) so we don't
                # have to remember to reset it to the null string when
                # we're done.

                $dpid = open (DUMPE2FS, "dumpe2fs @_[$dev] 2>/dev/null | grep −i journal |")
                    or die "Couldn't fork: $!\n";
                while (<DUMPE2FS>) {
#                   print "Dumpe2fs: $_";
                    $ext3{$_[$dev]} = "−j ";
                    last;
                }
                close (DUMPE2FS);

                if ($labels{@_[$dev]}) { # do we have a label?
                    $format .= "echo\necho formatting $checking@_[$dev]\n";
                    $format .= "mke2fs $ext3{@_[$dev]}\$blockcheck −L $labels{@_[$dev]} @_[$dev]\
                } else {
                    $format .= "echo\necho formatting $checking@_[$dev]\n";
                    $format .= "mke2fs $ext3{@_[$dev]}\$blockcheck @_[$dev]\n\n";
                }

                # logical Linux swap partition
            } elsif (@_[$parttype] == 82 ) {
                print OUTPUT "l\n@_[$firstcyl]\n";
                if (@_[$firstcyl] ne @_[$lastcyl]) {
                    print OUTPUT "@_[$lastcyl]\n";
                }
                print OUTPUT "t\n$partnumber\n82\n";
                $format .= "echo Making @_[$dev] a swap partition.\n";
                $format .= "mkswap \$blockcheck @_[$dev]\n\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" ) {
#               print ("Making DOS logical partition.\n");
                print ("dd if=@_[$dev] of=$outputfilepath$outputfilename$partnumber");
                print (" bs=512 count=1\n");
                system ("dd if=@_[$dev] of=$outputfilepath$outputfilename$partnumber bs=512 count
                print OUTPUT "l\n$partnumber\n@_[$firstcyl]\n";
                if (@_[$firstcyl] ne @_[$lastcyl]) { # in case it's all on one cylinder
                    print OUTPUT "@_[$lastcyl]\n";
                }
                print OUTPUT "t\n$partnumber\n@_[$parttype]\n";
                $format .= "echo\necho formatting $checking@_[$dev]\n";
                $format .= "mkdosfs \$blockcheck";
                if ( @_[$parttype] == b || @_[$parttype] == c) {
                    # 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$partnumber of=@_[$dev] bs=512 count=1\n\n";

            } else {
                # anything else partition
                print OUTPUT "l\n@_[$firstcyl]\n";
                if (@_[$firstcyl] ne @_[$lastcyl]) {
                    print OUTPUT "@_[$lastcyl]\n";
                }
                print OUTPUT "t\n$partnumber\n@_[$parttype]\n";
            }

        }

        # handle bootable partitions
        if (@_[$bootable] =~ /\*/) {
            print OUTPUT "a\n$partnumber\n";
        }
    }
}

print OUTPUT "v\nw\n";

close (OUTPUT);
close (FDISK);


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

print OUTPUT <<FINIS;

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 OUTPUT "dd if=/dev/zero of=$device bs=512 count=2\n\nsync\n\n";

print OUTPUT "fdisk $device \< $outputfilename\n\nsync\n\n";
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, "> ${outputfilepath}mount.$outputfilename")
    or die "Couldn't open output file ${outputfilepath}make.$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;

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.

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

print OUTPUT "\nmount | grep -i \"$device\"\n";

close (OUTPUT);

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

chmod 0700, "${outputfilepath}make.$outputfilename";
chmod 0700, "${outputfilepath}mount.$outputfilename";
chmod 0600, "${outputfilepath}$outputfilename";
```

11.1.2. `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` (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.,
# 59 Temple Place, Suite 330, Boston, MA 02111−1307 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

sync

fdisk /dev/hda < dev.hda

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"
```

11.1.3. `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.,
# 59 Temple Place, Suite 330, Boston, MA 02111-1307 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 /dev/hda3.
mkdir /target/
mount /dev/hda3 /target/

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

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

11.1.4. dev.hda

This data file is used at restore time. 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
t
1
6
a
1
n
p
2
30
44
n
e
3
45
```

```
1023
n
l
45
944
n
l
945
1023
t
6
82
v
w
```

11.1.5. `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.

```
#! /bin/sh

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

# Time−stamp: <2003−04−24 10:00:01 ccurley save.metadata>

# 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.,
# 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA

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


# 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/.


# 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
   local tarcmd="tar −cjf"      # The tar command.

   $tarcmd  $zip/$file.tar.bz2 $dirs # do it!!

   error=$?                     # Preserve the exit code

   if [ $error != 0 ]           # Did we fail?
   then                         # Yes
      echo "Tar failed with error $error"
      echo $tarcmd $zip/$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 zip="/mnt/zip";          # Where we will put archives
#  export save="/mnt/save";

RPMVABACKS=/etc                 # where we keep our backups
RPMVAROOT=rpmVa                 # The root name of the pg backups
ANC=${RPMVABACKS}/${RPMVAROOT}.anc      # name for the oldest (ancient) backup
OLD=${RPMVABACKS}/${RPMVAROOT}.old      # name for the middling oldest backup
NEW=${RPMVABACKS}/${RPMVAROOT}.txt      # name for the newest backup

if [ −f ${ANC} ]; then
echo "Deleting ${ANC}"
rm ${ANC}
fi
```

```
if [ −f ${OLD} ]; then
echo "Aging ${OLD}"
mv ${OLD} ${ANC}
fi

if [ −f ${NEW} ]; then
echo "Aging ${NEW}"
mv ${NEW} ${OLD}
fi


# 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"
make.fdisk /dev/hda

# back up RPM metadata

echo "Verifying RPMs."

rpm −Va | sort +2 −t ' ' > ${NEW}

echo "Finished verifying RPMs; now mounting the ZIP drive."

# Make sure we have the ZIP drive mounted.
umount $zip
modprobe ppa                    # Driver for 100MB parallel port ZIP disk
mount $zip                      # It should have ext2fs on partition 1.

# clean it all out
rm −r $zip/*
mkdir $zip/lost+found

echo "`hostname` bare metal ZIP disk, created `date`" > $zip/README.txt

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....

fdisk −l /dev/hda > $zip/fdisk.hda

ls −al /mnt > $zip/ls.mnt.txt
ls −al / > $zip/ls.root.txt
```

```
cd /

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

crunch boot boot
crunch root root --exclude boot/.cpan --exclude root/.mozilla
crunch etc etc --exclude etc/samba --exclude X11
crunch lib lib

crunch usr.sbin usr/sbin
crunch usr.bin usr/bin --exclude usr/bin/emacs --exclude usr/bin/emacs-21.2 --exclude usr/bin/ema
crunch sbin sbin
crunch bin bin
crunch dev dev
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.
crunch usr.lib usr/lib/*crack* usr/lib/libz* usr/lib/libssl* usr/lib/libcrypto*

# save the scripts we used to create the ZIP disk and the ones we will
# use to restore it.
mkdir $zip/root.bin
cp -p /root/bin/* $zip/root.bin
rm $zip/root.bin/*~ $zip/root.bin/#*#

echo "Testing our results."
find $zip -iname "*.bz2" | xargs bunzip2 -t

# Not a normal part of the process: we duplicate the ZIP disk onto an
# NFS mount elsewhere.

#  echo "Backing the ZIP drive to the NFS mount."

#  umount $save
#  mount $save

#  rm -r $save/zip
#  mkdir $save/zip
#  cp -pr $zip $save

du -hs ${zip}*
df -m
```

11.1.6. `restore.metadata`

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

```
#! /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.
# Time−stamp: <2003−04−24 09:59:26 ccurley restore.metadata>

# 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.,
# 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA

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

# 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/.


zip="/mnt";                     # Where we mount the zip drive.
target="/target";               # Where the hard drive to restore is mounted.

ls −lt $zip                     # Warm fuzzies for the user.

cd $target

# Restore the archived metadata files.
for archive in $( ls $zip/*.bz2 ); do
echo $archive
ls −al $archive
bzip2 −dc $archive | tar −xpf −
done

# 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.

for dir in mnt/zip mnt/cdrom mnt/floppy mnt/imports proc initrd; do
mkdir −p $target/$dir
done

chmod a−w $target/proc          # Restore /proc's read−only permissions

# Restore the scripts we used to create the ZIP disk and the ones we will
# use to restore it. These should be the latest & greatest in case we had
# to do any editing during 1st stage restore.
cp −p $zip/root.bin/* $target/root/bin
```

```
# Now install the boot sector.
chroot $target /sbin/lilo −C /etc/lilo.conf

df −m
```

11.1.7. `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/sh

# 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.

# Time−stamp: <2003−04−24 10:03:07 ccurley first.stage>

# 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.,
# 59 Temple Place, Suite 330, Boston, MA 02111−1307 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

cd /mnt/root.bin                # just in case we aren't already where we should be.

for drive in $( ls make.dev.* ); do
    ./$drive $blockcheck;
done
```

```
# 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.

# 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 $( ls −tr mount.dev.* ); do
    ./$drive;
done

./restore.metadata

# 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/sh

# 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.

# Time−stamp: <2003−04−24 09:56:05 ccurley back.up.all>

# 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.,
# 59 Temple Place, Suite 330, Boston, MA 02111−1307 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 /

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/sh

# 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.

# Time-stamp: <2003-04-24 09:56:52 ccurley back.up.all.ssh>

# 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.,
# 59 Temple Place, Suite 330, Boston, MA 02111-1307 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/sh

# A script to restore all of the data from an nfs mount. This is our final
# stage restore.

# Time-stamp: <2003-04-24 09:58:51 ccurley restore.all>

# 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.,
# 59 Temple Place, Suite 330, Boston, MA 02111-1307 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/sh

# 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.

# Time-stamp: <2003-04-24 09:59:10 ccurley restore.all.ssh>
# 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.,
# 59 Temple Place, Suite 330, Boston, MA 02111−1307 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. Rename them to `get.x` and `restore.x` where `x` is the name of the target computer. Edit them (the variable $target's initialization) to use the target computer's host name, or rewrite them to use a command line argument.

These scripts backup and restore the target completely, not just the stage one backup and restore. Also, note that `get.tester` backs up the ZIP disk as well, in case you need to replace a faulty ZIP disk.

I use these scripts routinely.

11.3.1. `get.tester`

```
#! /bin/sh

# 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.
# Time−stamp: <2003−04−24 09:57:14 ccurley get.tester>

# 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.,
# 59 Temple Place, Suite 330, Boston, MA 02111−1307 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/.

# 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.



# The host name of the computer to be backed up.
target=tester
zip=/mnt/zip

echo Backing up $target

echo Aging the ZIP disk backups.

rm −r $target.old.zip

mv $target.zip $target.old.zip

ssh $target "modprobe ppa ; mount −r $zip"

echo Copying the ZIP disk.

# −r for recursive copy, −p to preserve times and permissions, −q for
# quiet: no progress meter.

scp −qpr $target:$zip $target.zip

du −hs $target.zip


echo Aging the archives

rm $target.tar.old.bz2

mv $target.tar.bz2 $target.tar.old.bz2
```

```
echo Backing up $target to the backup server.

ssh $target tar -cf - / --exclude /mnt --exclude /proc --exclude /var/spool/squid\
    | bzip2 -9 | cat > $target.tar.bz2

echo Testing the results.
find . -iname "*.bz2" | xargs bunzip2 -t

ssh $target "eject $zip"
```

11.3.2. `restore.tester`

```
#! /bin/sh

# A script to restore all of the data to tester via ssh. This is our final
# stage restore.

# Time-stamp: <2003-04-24 09:59:45 ccurley restore.tester>

# 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.,
# 59 Temple Place, Suite 330, Boston, MA 02111-1307 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/.

# The host name of the computer to be restored.

target=tester

bunzip2 -dc $target.tar.bz2 | ssh $target "cd / ; tar -xpkf - "

ssh $target lilo
```

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.

- W. Curtis Preston's excellent *Unix Backup & Recovery*. This is the book that got me started on this bare metal recovery stuff. I highly recommend it; read my review.
- A list of small Linux disties.
- tomsrtbt, "The most Linux on 1 floppy disk." Tom also has links to other small disties.
- The Linux Documentation Project. See particularly the "*LILO, Linux Crash Rescue HOW−TO*"
- The Free Software Foundation's `parted` for editing (enlarging, shrinking, moving) partitions.
- Partition Image 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."
- Hugo Rabson's Mondo "... 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."
- Bacula is a GLPled backup product which has bare metal recovery code inspired in part by this HOWTO.
- The Crash Recovery Kit for Linux
- The Beeblebrox Project looks promising.
- "g4u ('ghost for unix') 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."

A. GNU Free Documentation License

Version 1.1, March 2000

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.

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

9. TERMINATION

10. FUTURE REVISIONS OF THIS LICENSE

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.