SquashFS HOWTO

Artemiy I. Pavlov

Sinevibes

Marco Cecchetti

<mrc (dot) ildp (dot) com>

Revision History

Revision 1.9	2008-07-24	Revised by:
Text corrections.		
Revision 1.8	2008-01-06	Revised by:
Changes according to SquashFS release 3.3. Some parts added.		
Revision 1.7	2005-03-25	Revised by:
Changes according to SquashFS release 2.1.		
Revision 1.6	2004-11-10	Revised by:
Changes according to SquashFS release 2.0. Text corrections.		
Revision 1.5	2004-06-07	Revised by:
Changes according to SquashFS release 2.0 alpha. Lots of description improvements and clarifications. Split instructions for Linux kernels of 2.6.x (new) and 2.4.x series.		
Revision 1.1	2004-05-22	Revised by:
Changes according to SquashFS release 1.3r3.		
Revision 1.0	2004-02-19	Revised by:
Initial Release, reviewed by LDP.		
Revision 0.2	2003-12-08	Revised by:
Text corrections, license added.		
Revision 0.1	2003-11-24	Revised by:
Initial version. Instructions for SquashFS release 1.3r2.		

Abstract

This HOWTO describes the usage of SquashFS - a highly-compressed read-only file system for Linux, which is intended for use in tiny-sized and embedded systems, and anywhere else you'd want to use a compressed file system. With this document, you'll learn how to prepare a SquashFS-ready Linux kernel, create a squashed file system and happily use it.

Home of this HOWTO

The SquashFS HOWTO lives at http://artemio.net/projects/linuxdoc/squashfs. There you will always find the latest version of the document, and will be able to send your feedback.

Table of Contents

1. What is SquashFS

1.1. Introduction

When creating tiny-sized and embedded Linux systems, every byte of the storage device (floppy, flash disk, etc.) is very important, so compression is used everywhere possible. Also, compressed file systems are frequently needed for archiving purposes. For huge public archives, as well as for personal media archives, this is essential.

SquashFS brings all this to a new level. It is a read-only file system that lets you compress whole file systems or single directories, write them to other devices/partitions or to ordinary files, and then mount them directly (if a device) or using a loopback device (if it is a file). The modular, compact system design of SquashFS is bliss. For archiving purposes, SquashFS gives you a lot more flexibility and performance speed than a tarball archive.

SquashFS is distributed as a Linux kernel source patch (which enables SquashFS read support in your kernel), the **mksquashfs** tool, which creates squashed file systems (in a file or on a block device) and the **unsquashfs** tool, which extracts multiple files from an existing squashed file system.

The latest SquashFS release tree is 3.x, the former one was 2.x. This document describes both these releases with proper notes given. For example, if some feature or parameter is different in these release trees, it will be written as follows: *new value (3.x) or old value (2.x)*

1.2. Overview of SquashFS

- Data, inodes and directories are compressed
- SquashFS stores full uid/gids (32 bits), and file creation time
- Files up to 2^64 bytes are supported; file systems can be up to 2^64 bytes
- Inode and directory data are highly compacted, and packed on byte boundaries; each compressed inode is on average 8 bytes in length (the exact length varies on file type, i.e. regular file, directory, symbolic link, and block/character device inodes have different sizes)
- SquashFS can use block sizes up to up to 64 Kb (2.x) and 1Mb (3.x). The default size is 128Kb (3.x), which achieves greater compression ratios than the normal 4K block size
- By the 2.x release it was introduced the concept of *fragment blocks*: an ability to join multiple files smaller than block size into a single block, achieving greater compression ratios
- File duplicates are detected and removed
- Both big and little endian architectures are supported; SquashFS can mount file systems created on different byte-order machines

---

1.3. Making it clear

Now let's make sure any further discussions will be clearer for you to understand. The procedure of getting SquashFS working basically consists of the following steps:

1. Patching and recompiling the target Linux kernel to enable SquashFS support
2. Compiling the **mksquashfs** and the **unsquashfs** tools
3. Creating a compressed file system with **mksquashfs**
4. Testing: mounting a squashed file system to a temporary location

5. Modifying the `/etc/fstab` or startup scripts of your target Linux system to mount the new squashed file system when needed

2. Getting ready for SquashFS

2.1. Acquiring SquashFS

The SquashFS home site is located at http://squashfs.sourceforge.net/ - it contains news for the latest release and it's changelog, as well as general information about SquashFS. You can grab the latest version at the SquashFS project page at SourceForge.

Squashfs is also avaliable with LZMA compression at http://www.squashfs-lzma.org/

2.2. Preparing a SquashFS-capable kernel

In order to read SquashFS, you need it supported in your kernel - just as if it was a `reiserfs` or `ext3` file system. You have to make sure there is an appropriate patch for your kernel version. It should be located in `kernel-patches/linux-2.x.y` subdirectory of the SquashFS source tree. Also, remember that in most cases you will need a *clean* (original) Linux kernel source from kernel.org. If your kernel source is from a distro vendor, it may be already pre-patched with custom vendor patches, and patching with a SquashFS patch will almost surely not work, as SquashFS patches are made against *original* Linux kernels. However some distributions make avaliable SquashFS-kernel-modules and SquashFS-tools packages from the repositories. If you consider to use these packages from your distro, you don't need to patch your kernel or to build SquashFS tools from source. In such case, make sure to get the proper module for your kernel from the distro repositories. Please note that doing so you will have an easy installation step, but no control on kernel configuration parameters if you mind to use SquashFS for particular purposes (eg. for embedded systems).

2.2.1. Patching the kernel source

With a kernel source and a proper SquashFS patch present, all you have to do is (we'll assume that you have your Linux kernel source in `/usr/src/linux` and that you have the SquashFS source in `/usr/src/squashfs`):

Change to the SquashFS source directory and copy the kernel patch (we'll assume it's named `squashfs-patch`) to `/usr/src/linux`.

```
        bash# cd /usr/src/squashfs
        bash# cp linux-2.x.y/squashfs-patch /usr/src/linux
```

Go to the linux kernel source directory `/usr/src/linux`:

```
        bash# cd /usr/src/linux
```

Note: please remember that we will not be leaving this directory during all further kernel-related procedures, and all paths will be given relative to `/usr/src/linux`.

Now patch the source with the SquashFS patch:

```
        bash# patch -p1 < squashfs-patch
```

2.2.2. Compiling a 2.6.x kernel

Cleanup and prepare the kernel source:

```
bash# make distclean
bash# make mrproper
```

Configure the kernel using your favourite method (config/menuconfig/xconfig/gconfig):

```
bash# make menuconfig
```

1. In the "*File systems*" section, "*Miscellaneous file systems*" subsection, enable the "*Squashed filesystem*" option, whether as module or bundled with the kernel. It is only obligatory to compile SquashFS inside the kernel if you plan using squashed initial RAM disks (**initrd**).
2. In the same subsection, **"do NOT"** enable the "*Additional option for memory-constrained system*", unless you are configuring your kernel for an embedded system.
3. If you would like to use a squashed initial RAM disk, enable the "*Initial RAM disk support*" in the "*Device drivers*" section, "*Block devices*" subsection.
4. If you want to be able to mount the squashed file system via a *loopback device* in future, you should enable "*Loopback device support*" in the "*Device drivers*" section, "*Block devices*" subsection.

Now you may compile the kernel and modules:

```
bash# make
```

2.2.3. Compiling a 2.4.x kernel

Configure the kernel:

```
bash# make menuconfig
```

1. In the "*File systems*" section, enable the "*Squashed filesystem*" option, whether as module or bundled with the kernel. It is only obligatory to compile SquashFS inside the kernel if you plan using squashed initial RAM disks (**initrd**).
2. If you would like to use a squashed initial RAM disk, enable the "*Initial RAM disk support*" in the "*Block devices*" section.
3. If you want to be able to mount the squashed file system via a *loopback device* in future, you should enable " *Loopback device support* " in the "*Block devices* " section.

Now you may compile the kernel and modules:

```
bash# make dep
bash# make bzImage
bash# make modules
```

2.2.4. Installing and testing the kernel

It's time to install your new SquashFS-enabled kernel. The instructions below are for installing and booting the kernel on the host machine. You may want to install and test it on the target system.

We assume that the kernel was compiled for a x86 architecture, and the compressed kernel image is located in the `arch/i386/boot/` subdirectory of the kernel tree. Now copy the kernel to the `/boot` directory (and name it `bzImage-sqsh` for convenience, if you like):

```
        bash# cp arch/i386/boot/bzImage /boot/bzImage-sqsh
```

Don't forget to install the kernel modules if you have any:

```
        bash# make modules_install
```

Modify your boot loader's configuration file to include your new kernel and install (update) the boot loader. Now you may reboot with your new kernel. When it boots, check that everything went fine:

```
        bash# cat /proc/filesystems
```

Or, if you built SquashFS support as a kernel module:

```
        bash# insmod squashfs
        bash# cat /proc/filesystems
```

If you see the `squashfs` line among other file systems, this means you have successfully enabled SquashFS in your kernel.

2.3. Compiling the SquashFS tools

Now you need to compile **mksquashfs** - the tool for creating squashed file systems and the **unsquashfs** which extracts files from an existing squashed file system.

```
        bash# cd /usr/src/squashfs/squashfs-tools
```

Compile and install the tools:

```
        bash# make
        bash# cp mksquashfs /usr/sbin
        bash# cp unsquashfs /usr/sbin
```

If everything went fine, typing **mksquashfs** or **unsquashfs** at the shell prompt should print the "usage" message.

2.4. Installing SquashFS on Debian

If you use Debian (or onother Linux distribution) you may want to get SquashFS module and tools from the distro repositories. With Debian you have to install the proper kernel module and the tools with the following commands:

(Assuming your architecture is x86)

```
        bash# apt-get install squashfs-modules-2.6-486  squashfs-tools
```

Now load the squashfs module for the Linux kernel and if it was correctly loaded you should find it in the relative list

```
        bash# modprobe squashfs

        bash# lsmod|grep squash
        squashfs               39620  0
```

Then you have to add the squashfs module to /etc/modules if you need it loaded at boot time.

```
        bash# echo squashfs >> /etc/modules
```

Note that at the time of writing, the Debian packages (Etch. 4.0 r2) relates to squashfs 3.1 release. Some latest options and features of 3.2 release may be not supported. Look at the next section for details.

3. The SquashFS tools exposed

3.1. Using mksquashfs

mksquashfs is the tool for creating new squashed file systems, and for appending new data to existing squashed file systems. The general command-line format for **mksquashfs** is:

```
bash# mksquashfs source1 source2 ... destination [options]
```

- `source1`, `source2`, etc.: files and directories to be added to the resulting file system, given with relative and/or absolute paths
- `destination`: a regular file (filesystem image file), or a block device (such as `/dev/fd0` or `/dev/hda3`) where you want to have your squashed file system

Notes for default **mksquashfs** behavior:

- When the new files are added to the new file system or appended to an existing one, **mksquashfs** will automatically rename files with duplicate names: if two or more files named `text` will appear in the same resulting directory, the second file will be renamed to `text_1`, third one to `text_2` and so on.
- Duplicate files will be removed, so there will be only one physical instance (By the SquashFS 2.x, you can disable the detection/removal of the duplicates with the **-no-duplicates** option).
- If *destination* has a pre-existing SquashFS file system on it, by default, the new *source* items will be appended to the existing root directory. Examine the options table below to force **mksquashfs** to overwrite the whole destination and/or change the way new source items are added.
- If a single source file or directory is given, it becomes the root in a newly created file system. If two or more source files and/or directories are given, they will all become sub-items in the root of the new file system.
- The resulting filesystem will be padded to a multiple of 4 Kb: this is required for filesystems to be used on block devices. If you are very sure you don't ned this, use the **-nopad** option to disable this operation.

See the next section for more details about all possible options.

3.2. Command-line options

All possible options for **mksquashfs** are shown in the table below.

Table 1. Command-line options of the mksquashfs tool

Option	Description
-2.0	force **mksquashfs** version 2.1 to create a version 2.0 filesystem
-all-root or **-root-owned**	make all files in the target file system owned by root (UID=0, GID=0)
-always-use-fragments	divide all files greater than block size into fragments (by the version 2.x). It will result in greater compression ratios
-b [block size]	use [block size] filesystem block size (32 Kbytes default for 2.x, 128 kbytes for 3.x) - this can be either 4096, 8192, 16384, 32768, 65536 or 131072

-be or **-le**	force a big or little endian file system, respectively
-check-data	enable additional file system checks
-e [file1] ([file2] ...)	specify which files and/or directories to omit from the new file system that is to be created
-ef [file]	specify a `file` which contains the list of files/directories to exclude
-force-gid [GID]	set all group IDs in target file system to [GID] (can be specified as a name or a number)
-force-uid [UID]	set all user IDs in target file system to [UID] (can be specified as a name or a number)
-info	print files, their original size and compression ratio, as they are added to the file system
-keep-as-directory	if the source is a single directory, force this directory to be a subdirectory of the root in the created file system
-noappend	if the destination file/device already contains a squashed file system, overwrite it, rather than append the new data to an existing file system
-no-duplicates	do not detect/remove duplicate file names
-noD or **-noDataCompression**	do not compress the data
-noF or **-noFragmentCompression**	do not compress the fragments (avaliable by 2.x)
-no-fragments	do not generate fragment blocks (avaliable by 2.x, this will produce almost the same filesystem as 1.x did)
-noI or **-noInodeCompression**	do not compress the inode table
-nopad	do not pad the resulting file system to a multiple of 4 KBytes
-root-becomes [name]	can be used while appending to a pre-existing squashed file system: it will make a new root, and [name] directory will contain all pre-existing files/directories
-version	print the version, copyright and license message
-recover [name]	recover filesystem data using recovery file [name] (3.3)
-no-recovery	don't create a recovery file (3.3).
-no-exports	don't make avaliable file system to export via NFS (3.x)
-no-sparse	don't check for sparse files (3.x)
-processors [number]	set the number of CPU to create file system. By default it will be used all avaliable processors (3.x)
--read-queue [size]	set input queue to [size] Mb. (Default is 64 Mb)(3.x)
-write-queue [size]	set output queue to [size] Mb (3.x)
-sort [sort_file]	sort files relating to priorities in [sort_file] (3.x)
-wildcards	enable the extended shell wildcards to exclude directories/files (to be used with -e)
-regex	enable to use POSIX regular expressions (3.3)

In most cases, you should leave all compression/block options by default, as they allow **mksquashfs** to achieve the best possible compression ratios.

3.3. Using unsquashfs

unsquashfs is the tool for extracting data from squashed file systems. The general command-line format for **unsquashfs** is:

```
unsquashfs [options] target [files/directories to extract]
```

- target is the squashed file system to extract.

Notes for **unsquashfs** behavior:

- By not specifying any *destination path*, unsquashfs extracts the compressed file system in the *./squashfs-root* directory.
- The tool does not extract a squashed file system on already exsisting directory unless the **-f** option is specified.
- You can specify on the command line, a multiple number of files/directories to extract and the items to be extracted can be also be given in a file with **-e [file]** option.

All possible options for **unsquashfs** are shown in the table below.

Table 2. Command-line options of the unsquashfs tool

Option	Description
-v[ersion]	print the version, licence and copyright message
-i[nfo]	print the files as they are extracted from the file system
-l[ist]	list the squashed file system without extracting files
-li	list files with theyr attributes as they are unsquashed (3.3)
-ll	list the squashed file system files with attributes without any extraction (3.3)
-d[estination] path	specify a destination path for unsquashed items
-f[orce]	if files exist overwrite them
-s[tat]	display file system's superblock informations (it can discover the file system version and the options used to compress it) (3.3)
-e[f] [extract file]	list of directories or files to extract (entries given one per line) (3.3)
-r[egex]	treat extract names as POSIX regular expressions (3.3)

Note that by 3.x release you can extract 1.x and 2.x squashed file system too.

4. Creating and using squashed file systems

4.1. Basic steps

In order to create a squashed file system out of a single directory (say, `/some/dir`), and output it to a regular file (thus, producing a file system image), you need to say only one magic phrase:

```
bash# mksquashfs /some/dir dir.sqsh
```

mksquashfs will perform the squashing and print the resulting number of inodes and size of data written, as well as the average compression ratio. Now you have your `/some/dir` directory image in the `dir.sqsh` file. You can now use the **mount** command to mount it using a loopback device:

```
bash# mkdir /mnt/dir
bash# mount dir.sqsh /mnt/dir -t squashfs -o loop
```

To check if you have what's expected:

```
bash# ls /mnt/dir
```

If you want to output the file system directly into a device (say, your floppy at `/dev/fd0`):

```
bash# mksquashfs /some/dir /dev/fd0
```

Then just **mount** the device:

```
bash# mount /dev/fd0 /mnt/floppy -t squashfs
```

And check if it's okay:

```
bash# ls /mnt/floppy
```

4.2. Squashing file systems

Operations described here correspond to most cases where a read-only compressed file system can be used, whether you want it to be on a block device or in a file. This could be anything from large FTP/HTTP-served archives that don't change often, to having a squashed `/usr` partition and anything alike with these.

4.2.1. Example 1

Let's suppose you have a `/var/arch` directory with lots of files and that you want to turn it into a squashed file system and keep it on your root partition as a file (it will be a file system image that you will mount via a loopback device). The operations needed to perform are as follows.

Squash the directory, then mount it via loopback to test it:

```
bash# mksquashfs /var/arch /var/arch.sqsh
bash# mkdir /mnt/tmp
bash# mount /var/arch.sqsh /mnt/tmp -t squashfs -o loop
bash# ls /mnt/tmp
```

If everything is as expected, make this file system mount automatically at boot time by adding this line to your `/etc/fstab`:

```
        /var/arch.sqsh  /var/arch       squashfs        ro,defaults     0 0
```

Unmount the file system from the temporary mount point, and mount using it's fstab entry:

```
        bash# umount /mnt/tmp
        bash# mount /var/arch
```

Now just ensure that everything works fine:

```
        bash# ls /var/arch
```

4.2.2. Example 2

Say you have two hard disk partitions, `/dev/hda6` (which is empty) and `/dev/hda7` (which is bigger than `/dev/hda6`, mounted at `/var/arch`, contains some data and is full). Now, say you want to squash the `/dev/hda7` file system and move it to `/dev/hda6`, then use `/dev/hda7` for some other purposes. We will suppose you have the following line in `/etc/fstab` (**reiserfs** is just an example file system used on `/dev/hda7`):

```
        /dev/hda7       /var/arch       reiserfs        defaults        0 0
```

In the same fashion as with the previous example:

```
        bash# mksquashfs /var/arch /var/arch.sqsh
        bash# mkdir /mnt/tmp
        bash# mount /var/arch.sqsh /mnt/tmp -t squashfs -o loop
        bash# ls /mnt/tmp
```

If everything went fine, unmount `/dev/hda7` (if needed) and use **dd** to copy `/var/arch.sqsh` to `/dev/hda6`:

```
        bash# umount /dev/hda7
        bash# dd if=/var/arch.sqsh of=/dev/hda6
```

Now change the line in `/etc/fstab` for `/dev/hda7` to:

```
        /dev/hda6       /var/arch       squashfs        ro,defaults     0 0
```

Mount the new file system and check to see if all went fine:

```
        bash# mount /var/arch
        bash# ls /var/arch
```

Don't forget to erase the unneeded file system image:

```
        bash# rm /var/arch.sqsh
```

4.3. Creating tiny/embedded systems

By saying "tiny/embedded", I mean Linux systems that are being built for booting from floppy disks, IDE/USB flash disks, iso9660 CD-ROMs, small-sized hard drives and the like. Whether you want to have your whole root file system on a single media (a single partition, a single floppy), or have a modular system (several floppies or disk partitions), the procedure is almost identical. Creating such Linux systems themselves is out of scope of this HOWTO - there are dedicated HOWTOs and guides for this (like the *Bootdisk HOWTO* and *Linux From Scratch* - visit www.tldp.org to retrieve these documents).

4.3.1. Squashed file systems on floppy/flash/hard disks

In order to use SquashFS for creating Linux systems on small disks, you just have to follow the usual steps for creating a minimal system, performing the following operations at respective points:

1. When developing a kernel for your system, make sure you enable SquashFS support so it can mount squashed file systems
2. Use **mksquashfs** for creating read-only initial ram disks and/or root and/or other file systems
3. Don't forget to set file system types to `squashfs` in `/etc/fstab` and/or the startup scripts of your system for mounting squashed file systems

Floppy example. Let's say you have your floppy system tree at `/home/user/floppylinux` and you want to place the root file system on one floppy and `/usr` on another. What you should do is:

```
bash# cd /home/user
bash# mksquashfs floppylinux root.sqsh -e usr
bash# mksquashfs floppylinux/usr usr.sqsh
```

Note 1: you can see here how we use the **-e** option to exclude the `/usr` directory for root file system's image.

Note 2: don't forget to specify `squashfs` in your root disk's `/etc/fstab` or startup scripts when mounting the `/usr` file system.

Insert a root disk in your 3.5" floppy drive (I assume you have a lilo or grub on it, and, thus, a file system exists on this floppy, and the root file system will reside under the `/boot` directory of this file system):

```
bash# mount /mnt/floppy
bash# cp root.sqsh /mnt/floppy/boot
```

When done, unmount the root floppy, change the floppy to a `/usr` disk and use **dd** to transfer the `usr` file system:

```
bash# dd if=usr.sqsh of=/dev/fd0
```

4.3.2. Squashed file systems on CD-ROMs

With SquashFS, you can compress large file systems that will be used in live CDs (just as an example). For this purpose SquashFS is also used with UnionFS.

1. Enable SquashFS in the linux kernel of the target system
2. Create a squashed root file system
3. Modify the `/etc/fstab` or startup scripts of the target system to mount the squashd file system when you need it

If you create a root file system out of a running Linux system, use the **-e** option for **mksquashfs** to exclude all pseudo-filesystems such as `/proc`, `/sys` (on linux kernels after 2.5.x) and `/dev` (when using DevFS). Also, don't forget to add the file system image itself that is being created with **mksquashfs** (I think you know the reasons for these exclusions).

4.4. Making it writeble

As mentioned, another interesting use for **SquashFS** is with **Unionfs** filesystem, which provides *copy-on-write* semantics for the read-only file systems, enahancing the possibilities. (For unionfs you can look at http://www.filesystems.org/project-unionfs.html)

Just to make an example, you may want to make your /home/user squashed, to compress and backup your files without losing the possibility to apply changes or writing new files.

Create the `ro.fs` squashed file system and the `rw.fs` dir.

```
bash# mksquashfs /home/user1 ro.fs
bash# mkdir /home/rw.fs
```

Mount the squashed ro.fs file system using the loopback device

```
bash# mount -t squashfs ro.fs /mnt -o loop
```

mount the unionfs file system, that makes `/mnt` and `/home/rw.fs` apparently merged under `/home/user1` location.

```
bash# cd /home
bash# mount -t unionfs -o dirs=rw.fs=rw:/mnt=ro unionfs user1
```

As you can see, now you can create new files in /home/user1.

```
bash# cd /home/user1
bash# touch file1
bash# ls
```

umount the unionfs and the squashfs file systems and list the content of /home/user1 and /home/rw.fs dir

```
bash# cd ..
bash# umount /home/user1
bash# umount /mnt

bash# ls /home/user1
bash# ls /home/rw.fs
```

You can see that the new `file1` was created in `/home/rw.fs`

When you want to add the new created files to the *stable* and *compressed* squashed file system, you have to add them to the exsisting one.

```
bash# mksquashfs /home/rw.fs /home/ro.fs
```

Now, to mount your squashed user home directory at system startup, you can do as follow:

Make squashfs and unionfs modules loaded at boot time.

```
bash# echo squashfs >> /etc/modules
bash# echo unionfs >> /etc/modules
```

Change the owner of the writeble branch to match user1.

```
chown user1 /home/rw.fs
```

Add these lines to /etc/fstab file to mount squashfs and unionfs at boot time.

```
...
/home/ro.fs  /mnt squashfs loop 0 0
unionfs /home/user1 unionfs dirs=/home/rw.fs=rw:/mnt=ro 0 0
```

5. Acknowledgements

I would like to express my sincere thanks and immeasurable respect to:

- Phillip Lougher - for his brilliant work under squashfs, for creating an exculsive patch for linux-2.4.18, for his help with polishing this howto and answers to my mails
- Tabatha Marshall at TLDP for helping me with bringing this HOWTO to the final 1.0 release
- Everybody at The Linux Documentation Project for their great work under all the HOWTOs and guides that helped me a lot with exploring and hacking Linux
- All those at the TLDP mailing lists who helped me with getting started
- Endless thanks and respect to everybody who develops open-source software

Artemiy I. Pavlov

I want to thank Artemiy for his patience answering my emails, allowing me to work to this howto. I would express also my thanks and respect to everybody who is involved in free software

Marco Cecchetti

6. License

This document may be used and distributed under the terms and conditions set forth in the Open Content licence. In short, this means that you can freely modify and re-distribute the HOWTO under the main condition that you keep the author and copyright the article along. The full text of the licence is available at http://www.opencontent.org/opl.shtml