# SquashFS HOWTO


## Artemiy I. Pavlov

ArtemioLabs

<ap (at) artemio (dot) net>


2005−03−25

**Revision History**

| | |
|---|---|
| Revision 1.7 | 2005−03−25 |
| Changes according to SquashFS release 2.1. | |
| Revision 1.6 | 2004−11−10 |
| Changes according to SquashFS release 2.0. Text corrections. | |
| Revision 1.5 | 2004−06−07 |
| 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 |
| Changes according to SquashFS release 1.3r3. | |
| Revision 1.0 | 2004−02−19 |
| Initial Release, reviewed by LDP. | |
| Revision 0.2 | 2003−12−08 |
| Text corrections, license added. | |
| Revision 0.1 | 2003−11−24 |
| 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 sqaushed 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**
1.1. Introduction....1
1.2. Overview of SquashFS....1
1.3. Making it clear....1

**2. Getting ready for SquashFS....3**
2.1. Acquiring SquashFS....3
2.2. Preparing a SquashFS−capable kernel....3
2.2.1. Patching the kernel source....3
2.2.2. Compiling a 2.6.x kernel....3
2.2.3. Compiling a 2.4.x kernel....4
2.2.4. Installing and testing the kernel....4
2.3. Compiling the mksquashfs tool....5

**3. The mksquashfs tool, exposed....6**
3.1. Using mksquashfs....6
3.2. Command−line options....6

**4. Creating and using squashed file systems....8**
4.1. Basic steps....8
4.2. Squashing file systems....8
4.2.1. Example 1....8
4.2.2. Example 2....9
4.3. Creating tiny/embedded systems....9
4.3.1. Squashed file systems on floppy/flash/hard disks....10
4.3.2. Squashed file systems on CD−ROMs....10

**5. Acknowledgements....11**

**6. License....12**

# 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 .tar.gz archive.

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

The latest SquashFS release tree is 2.x, the former one was 1.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 (2.x) or old value (1.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 2^32 bytes are supported; file systems can be up to 2^32 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 32 Kb (1.x) and 64Kb (2.x), which achieves greater compression ratios than the normal 4K block size
- SquashFS 2.x inroduced 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 fro 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**tool
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 SqaushFS project page at SourceForge.

## 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 `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.

### 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. 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.
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 "*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 mksquashfs tool

Now you need to compile **mksquashfs –** the tool for creating squashed file systems.

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

Compile and install **mksquashfs**:

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

If everything went fine, typing **mksquashfs** at the shell prompt should print it's "usage" message.

# 3. The mksquashfs tool, 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 filke 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 (with SquashFS 2.x, you can disable the detection/rtemoval 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. Please note that it is not possible to append to a file system created with **mksquashfs**1.x using **mksquashfs**2.x. You will need to mount the SquashFS−1.x file system and copy the files to some location, and then join them with other needed files to create a SquashFS−2.x file system.
- 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 (2.x only, will result in greater compression ratios) |
| **–b [block size]** | use [block size] filesystem block size (32 Kbytes default) – this can be either 512, 1024, 2048, 4096, 8192, 16384 or 32768 |
| **–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 (2.x only) |
| **–no–fragments** | do not generate fragment blocks (2.x only, 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 |

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

---

# 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).

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).

---

# 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 Projectfor 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

---

# 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