NAME

guestfs−faq − libguestfs Frequently Asked Questions (FAQ)

ABOUT LIBGUESTFS

What is libguestfs?

libguestfs is a way to create, access and modify disk images. You can look inside disk images, modify the files they contain, create them from scratch, resize them, and much more. It’s especially useful from scripts and programs and from the command line.

libguestfs is a C library (hence ‘‘lib−’’), and a set of tools built on this library, and bindings for many common programming languages.

For more information about what libguestfs can do read the introduction on the home page (http://libguestfs.org).

What are the virt tools?

Virt tools (website: http://virt−tools.org) are a whole set of virtualization management tools aimed at system administrators. Some of them come from libguestfs, some from libvirt and many others from other open source projects. So virt tools is a superset of libguestfs. However libguestfs comes with many important tools. See http://libguestfs.org for a full list.

Does libguestfs need { libvirt / KVM / Red Hat / Fedora }?

No!

libvirt is not a requirement for libguestfs.

libguestfs works with any disk image, including ones created in VMware, KVM, qemu, VirtualBox, Xen, and many other hypervisors, and ones which you have created from scratch.

RedHat sponsors (ie. pays for) development of libguestfs and a huge number of other open source projects. But you can run libguestfs and the virt tools on many different Linux distros and Mac OS X. We try our best to support all Linux distros as first-class citizens. Some virt tools have been ported to Windows.

How does libguestfs compare to other tools?

vs. kpartx

Libguestfs takes a different approach from kpartx. kpartx needs root, and mounts filesystems on the host kernel (which can be insecure − see **guestfs−security**(1)). Libguestfs isolates your host kernel from guests, is more flexible, scriptable, supports LVM, doesn’t require root, is isolated from other processes, and cleans up after itself. Libguestfs is more than just file access because you can use it to create images from scratch.

vs. vdfuse

vdfuse is like kpartx but for VirtualBox images. See the kpartx comparison above. You can use libguestfs on the partition files exposed by vdfuse, although it’s not necessary since libguestfs can access VirtualBox images directly.

vs. qemu-nbd

NBD (Network Block Device) is a protocol for exporting block devices over the network. qemu-nbd is an NBD server which can handle any disk format supported by qemu (eg. raw, qcow2). You can use libguestfs and qemu-nbd or nbdkit together to access block devices over the network, for example: `guestfish −a nbd://remote`

vs. mounting filesystems in the host

Mounting guest filesystems in the host is insecure and should be avoided completely for untrusted guests. Use libguestfs to provide a layer of protection against filesystem exploits. See also **guestmount**(1).

vs. parted

Libguestfs supports LVM. Libguestfs uses parted and provides most parted features through the libguestfs API.

GETTING HELP AND REPORTING BUGS

How do I know what version I'm using?

The simplest method is:

```
guestfish --version
```

Libguestfs development happens along an unstable branch and we periodically create a stable branch which we backport stable patches to. To find out more, read "LIBGUESTFS VERSION NUMBERS" in **guestfs**(3).

How can I get help?
What mailing lists or chat rooms are available?

If you are a RedHat customer using Red Hat Enterprise Linux, please contact RedHatSupport: http://redhat.com/support

There is a mailing list, mainly for development, but users are also welcome to ask questions about libguestfs and the virt tools: https://www.redhat.com/mailman/listinfo/libguestfs

You can also talk to us on IRC channel `#guestfs` on Libera Chat. We're not always around, so please stay in the channel after asking your question and someone will get back to you.

For other virt tools (not ones supplied with libguestfs) there is a general virt tools mailing list: https://www.redhat.com/mailman/listinfo/virt−tools−list

How do I report bugs?

Please use the following link to enter a bug in Bugzilla:

https://bugzilla.redhat.com/enter_bug.cgi?component=libguestfs&product=Virtualization+Tools

Include as much detail as you can and a way to reproduce the problem.

Include the full output of **libguestfs−test−tool**(1).

COMMON PROBLEMS

See also "LIBGUESTFS GOTCHAS" in **guestfs**(3) for some "gotchas" with using the libguestfs API.

"Could not allocate dynamic translator buffer"

This obscure error is in fact an SELinux failure. You have to enable the following SELinux boolean:

```
setsebool -P virt_use_execmem=on
```

For more information see https://bugzilla.redhat.com/show_bug.cgi?id=806106.

"child process died unexpectedly"

[This error message was changed in libguestfs 1.21.18 to something more explanatory.]

This error indicates that qemu failed or the host kernel could not boot. To get further information about the failure, you have to run:

```
libguestfs-test-tool
```

If, after using this, you still don't understand the failure, contact us (see previous section).

libguestfs: error: cannot find any suitable libguestfs supermin, fixed or old-style appliance on LIBGUESTFS_PATH
febootstrap-supermin-helper: ext2: parent directory not found
supermin-helper: ext2: parent directory not found

[This issue is fixed permanently in libguestfs ≥ 1.26.]

If you see any of these errors on Debian/Ubuntu, you need to run the following command:

```
sudo update-guestfs-appliance
```

"Permission denied" when running libguestfs as root

You get a permission denied error when opening a disk image, even though you are running libguestfs as root.

This is caused by libvirt, and so only happens when using the libvirt backend. When run as root, libvirt decides to run the qemu appliance as user `qemu.qemu`. Unfortunately this usually means that qemu cannot open disk images, especially if those disk images are owned by root, or are present in directories which require root access.

There is a bug open against libvirt to fix this: https://bugzilla.redhat.com/show_bug.cgi?id=1045069

You can work around this by one of the following methods:

- Switch to the direct backend:

  ```
  export LIBGUESTFS_BACKEND=direct
  ```

- Don't run libguestfs as root.
- Chmod the disk image and any parent directories so that the qemu user can access them.
- (Nasty) Edit */etc/libvirt/qemu.conf* and change the `user` setting.

execl: /init: Permission denied

Note: If this error happens when you are using a distro package of libguestfs (eg. from Fedora, Debian, etc) then file a bug against the distro. This is not an error which normal users should ever see if the distro package has been prepared correctly.

This error happens during the supermin boot phase of starting the appliance:

```
supermin: mounting new root on /root
supermin: chroot
execl: /init: Permission denied
supermin: debug: listing directory /
[...followed by a lot of debug output...]
```

This is a complicated bug related to **supermin**(1) appliances. The appliance is constructed by copying files like */bin/bash* and many libraries from the host. The file `hostfiles` lists the files that should be copied from the host into the appliance. If some files don't exist on the host then they are missed out, but if these files are needed in order to (eg) run */bin/bash* then you'll see the above error.

Diagnosing the problem involves studying the libraries needed by */bin/bash*, ie:

```
ldd /bin/bash
```

comparing that with `hostfiles`, with the files actually available in the host filesystem, and with the debug output printed in the error message. Once you've worked out which file is missing, install that file using your package manager and try again.

You should also check that files like */init* and */bin/bash* (in the appliance) are executable. The debug output shows file modes.

DOWNLOADING, INSTALLING, COMPILING LIBGUESTFS

Where can I get the latest binaries for ...?

Fedora ≥ 11

Use:

```
yum install '*guestf*'
```

For the latest builds, see: http://koji.fedoraproject.org/koji/packageinfo?packageID=8391

Red Hat Enterprise Linux

RHEL 6

RHEL 7

It is part of the default install. On RHEL 6 and 7 (only) you have to install `libguestfs-winsupport` to get Windows guest support.

Debian and Ubuntu

For libguestfs < 1.26, after installing libguestfs you need to do:

```
sudo update-guestfs-appliance
```

(This script has been removed on Debian/Ubuntu with libguestfs ≥ 1.26 and instead the appliance is built on demand.)

On Ubuntu only:

```
sudo chmod 0644 /boot/vmlinuz*
```

You may need to add yourself to the `kvm` group:

```
sudo usermod −a −G kvm yourlogin
```

Debian Squeeze (6)
: Hilko Bengen has built libguestfs in squeeze backports: http://packages.debian.org/search?keywords=guestfs&searchon=names&section=all&suite=squeeze−backports

Debian Wheezy and later (7+)
: Hilko Bengen supports libguestfs on Debian. Official Debian packages are available: http://packages.debian.org/search?keywords=libguestfs

Ubuntu
: We don't have a full time Ubuntu maintainer, and the packages supplied by Canonical (which are outside our control) are sometimes broken.

Canonical decided to change the permissions on the kernel so that it's not readable except by root. This is completely stupid, but they won't change it (https://bugs.launchpad.net/ubuntu/+source/linux/+bug/759725). So every user should do this:

```
sudo chmod 0644 /boot/vmlinuz*
```

Ubuntu 12.04
: libguestfs in this version of Ubuntu works, but you need to update febootstrap and seabios to the latest versions.

You need febootstrap ≥ 3.14−2 from: http://packages.ubuntu.com/precise/febootstrap

After installing or updating febootstrap, rebuild the appliance:

```
sudo update−guestfs−appliance
```

You need seabios≥ 0.6.2−0ub untu2.1 or ≥ 0.6.2−0ubuntu3 from: http://packages.ubuntu.com/precise−updates/seabios or http://packages.ubuntu.com/quantal/seabios

Also you need to do (see above):

```
sudo chmod 0644 /boot/vmlinuz*
```

Gentoo
: Libguestfs was added to Gentoo in 2012−07 by Andreis Vinogradovs (libguestfs) and Maxim Koltsov (mainly hivex). Do:

```
emerge libguestfs
```

Mageia
: Libguestfs was added to Mageia in 2013−08. Do:

```
urpmi libguestfs
```

SuSE
: Libguestfs was added to SuSE in 2012 by Olaf Hering.

ArchLinux
: Libguestfs was added to the AUR in 2010.

Other Linux distro
: Compile from source (next section).

Other non-Linux distro
You'll have to compile from source, and port it.

How can I compile and install libguestfs from source?

You can compile libguestfs from git or a source tarball. Read the README file before starting.

Git: https://github.com/libguestfs/libguestfs Source tarballs: http://libguestfs.org/download

Don't run `make install`! Use the `./run` script instead (see README).

How can I compile and install libguestfs if my distro doesn't have new enough qemu/supermin/kernel?

Libguestfs needs supermin 5. If supermin 5 hasn't been ported to your distro, then see the question below.

First compile qemu, supermin and/or the kernel from source. You do *not* need to `make install` them.

In the libguestfs source directory, create two files. `localconfigure` should contain:

```
source localenv
#export PATH=/tmp/qemu/x86_64-softmmu:$PATH
./configure --prefix /usr "$@"
```

Make `localconfigure` executable.

`localenv` should contain:

```
#export SUPERMIN=/tmp/supermin/src/supermin
#export LIBGUESTFS_HV=/tmp/qemu/x86_64-softmmu/qemu-system-x86_64
#export SUPERMIN_KERNEL=/tmp/linux/arch/x86/boot/bzImage
#export SUPERMIN_KERNEL_VERSION=4.XX.0
#export SUPERMIN_MODULES=/tmp/lib/modules/4.XX.0
```

Uncomment and adjust these lines as required to use the alternate programs you have compiled.

Use `./localconfigure` instead of `./configure`, but otherwise you compile libguestfs as usual.

Don't run `make install`! Use the `./run` script instead (see README).

How can I compile and install libguestfs without supermin?

If supermin 5 supports your distro, but you don't happen to have a new enough supermin installed, then see the previous question.

If supermin 5 doesn't support your distro at all, you will need to use the "fixed appliance method" where you use a pre-compiled binary appliance. To build libguestfs without supermin, you need to pass `--disable-appliance --disable-daemon` to either *./configur e* or *./configure* (depending whether you are building respectively from git or from tarballs). Then, when using libguestfs, you **must** set the `LIBGUESTFS_PATH` environment variable to the directory of a pre-compiled appliance, as also described in "FIXED APPLIANCE" in **guestfs−internals**(1).

For pre-compiled appliances, see also: http://libguestfs.org/download/binaries/appliance/.

Patches to port supermin to more Linux distros are welcome.

How can I add support for sVirt?

Note for Fedora/RHEL users: This configuration is the default starting with Fedora18 and RHEL7. If you find any problems, please let us know or file a bug.

SVirt provides a hardened appliance using SELinux, making it very hard for a rogue disk image to "escape" from the confinement of libguestfs and damage the host (it's fair to say that even in standard libguestfs this would be hard, but sVirt provides an extra layer of protection for the host and more importantly protects virtual machines on the same host from each other).

Currently to enable sVirt you will need libvirt ≥ 0.10.2 (1.0 or later preferred), libguestfs ≥ 1.20, and the SELinux policies from recent Fedora. If you are not running Fedora18+, you will need to make changes to your SELinux policy − contact us on the mailing list.

Once you have the requirements, do:

```
./configure --with-default-backend=libvirt       # libguestfs >= 1.22
./configure --with-default-attach-method=libvirt # libguestfs <= 1.20
make
```

Set SELinux to Enforcing mode, and sVirt should be used automatically.

All, or almost all, features of libguestfs should work under sVirt. There is one known shortcoming: **virt−rescue** (1) will not use libvirt (hence sVirt), but falls back to direct launch of qemu. So you won't currently get the benefit of sVirt protection when using virt-rescue.

You can check if sVirt is being used by enabling libvirtd logging (see */etc/libvirt/libvirtd.log*), killing and restarting libvirtd, and checking the log files for ''SettingSELinuxcontexton...'' messages.

In theory sVirt should support AppArmor, but we have not tried it. It will almost certainly require patching libvirt and writing an AppArmor policy.

Libguestfs has a really long list of dependencies!

The base library doesn't depend on very much, but there are three causes of the long list of other dependencies:

1. Libguestfs has to be able to read and edit many different disk formats. For example, XFS support requires XFS tools.
2. There are language bindings for many different languages, all requiring their own development tools. All language bindings (except C) are optional.
3. There are some optional library features which can be disabled.

Since libguestfs ≥ 1.26 it is possible to split up the appliance dependencies (item 1 in the list above) and thus have (eg) `libguestfs−xfs` as a separate subpackage for processing XFS disk images. We encourage downstream packagers to start splitting the base libguestfs package into smaller subpackages.

Errors during launch on Fedora ≥ 18, RHEL ≥ 7

In Fedora ≥ 18 and RHEL ≥ 7, libguestfs uses libvirt to manage the appliance. Previously (and upstream) libguestfs runs qemu directly:

```
│ libguestfs                         │

│ direct backend │ libvirt backend │

       ↓                  ↓

   │ qemu  │         │ libvirtd │

                          ↓

                      │ qemu  │


   upstream          Fedora 18+
   non−Fedora         RHEL 7+
   non−RHEL
```

The libvirt backend is more sophisticated, supporting SELinux/sVirt (see above), hotplugging and more. It is, however, more complex and so less robust.

If you have permissions problems using the libvirt backend, you can switch to the direct backend by setting this environment variable:

```
export LIBGUESTFS_BACKEND=direct
```

before running any libguestfs program or virt tool.

How can I switch to a fixed / prebuilt appliance?

This may improve the stability and performance of libguestfs on Fedora and RHEL.

Any time after installing libguestfs, run the following commands as root:

```
mkdir -p /usr/local/lib/guestfs/appliance
libguestfs-make-fixed-appliance /usr/local/lib/guestfs/appliance
ls -l /usr/local/lib/guestfs/appliance
```

Now set the following environment variable before using libguestfs or any virt tool:

```
export LIBGUESTFS_PATH=/usr/local/lib/guestfs/appliance
```

Of course you can change the path to any directory you want. You can share the appliance across machines that have the same architecture (eg. all x86−64), but note that libvirt will prevent you from sharing the appliance across NFS because of permissions problems (so either switch to the direct backend or don't use NFS).

How can I speed up libguestfs builds?

By far the most important thing you can do is to install and properly configure Squid. Note that the default configuration that ships with Squid is rubbish, so configuring it is not optional.

A very good place to start with Squid configuration is here: https://fedoraproject.org/wiki/Extras/MockTricks#Using_Squid_to_Speed_Up_Mock_package_downloads

Make sure Squid is running, and that the environment variables `$http_proxy` and `$ftp_proxy` are pointing to it.

With Squid running and correctly configured, appliance builds should be reduced to a few minutes.

How can I speed up libguestfs builds (Debian)?

Hilko Bengen suggests using "approx" which is a Debian archive proxy (http://packages.debian.org/approx). This tool is documented on Debian in the**appr ox**(8) manual page.

SPEED, DISK SPACE USED BY LIBGUESTFS

Note: Most of the information in this section has moved: **guestfs−performance**(1).

Upload or write seem very slow.

If the underlying disk is not fully allocated (eg. sparse raw or qcow2) then writes can be slow because the host operating system has to do costly disk allocations while you are writing. The solution is to use a fully allocated format instead, ie. non-sparse raw, or qcow2 with the `preallocation=metadata` option.

Libguestfs uses too much disk space!

libguestfs caches a large-ish appliance in:

```
/var/tmp/.guestfs−<UID>
```

If the environment variable `TMPDIR` is defined, then *$TMPDIR/.guestfs−<UID>* is used instead.

It is safe to delete this directory when you are not using libguestfs.

virt-sparsify seems to make the image grow to the full size of the virtual disk

If the input to **virt−sparsify**(1) is raw, then the output will be raw sparse. Make sure you are measuring the output with a tool which understands sparseness such as `du −sh`. It can make a huge difference:

```
$ ls −lh test1.img
−rw−rw−r−−. 1 rjones rjones 100M Aug  8 08:08 test1.img
$ du −sh test1.img
3.6M   test1.img
```

(Compare the apparent size **100M** vs the actual size **3.6M**)

If all this confuses you, use a non-sparse output format by specifying the −−*convert* option, eg:

```
virt−sparsify −−convert qcow2 disk.raw disk.qcow2
```

Why doesn't virt-resize work on the disk image in-place?

Resizing a disk image is very tricky — especially making sure that you don't lose data or break the bootloader. The current method effectively creates a new disk image and copies the data plus bootloader from the old one. If something goes wrong, you can always go back to the original.

If we were to make virt-resize work in-place then there would have to be limitations: for example, you wouldn't be allowed to move existing partitions (because moving data across the same disk is most likely to corrupt data in the event of a power failure or crash), and LVM would be very difficult to support (because of the almost arbitrary mapping between LV content and underlying disk blocks).

Another method we have considered is to place a snapshot over the original disk image, so that the original data is untouched and only differences are recorded in the snapshot. You can do this today using `qemu−img create`+ `virt−resize`, b ut qemu currently isn't smart enough to recognize when the same block is written back to the snapshot as already exists in the backing disk, so you will find that this doesn't save you any space or time.

In summary, this is a hard problem, and what we have now mostly works so we are reluctant to change it.

Why doesn't virt-sparsify work on the disk image in-place?

In libguestfs ≥ 1.26, virt-sparsify can now work on disk images in place. Use:

```
 virt−sparsify −−in−place disk.img
```

But first you should read ''IN-PLACE SPARSIFICATION'' in **virt−sparsify** (1).

PROBLEMS OPENING DISK IMAGES

Remote libvirt guests cannot be opened.

Opening remote libvirt guests is not supported at this time. For example this won't work:

```
 guestfish −c qemu://remote/system −d Guest
```

To open remote disks you have to export them somehow, then connect to the export. For example if you decided to use NBD:

```
 remote$ qemu−nbd −t −p 10809 guest.img
  local$ guestfish −a nbd://remote:10809 −i
```

Other possibilities include ssh (if qemu is recent enough), NFS or iSCSI. See ''REMOTE STORAGE'' in **guestfs** (3).

How can I open this strange disk source?

You have a disk image located inside another system that requires access via a library / HTTP / REST / proprietary API, or is compressed or archived in some way. (One example would be remote access to OpenStack glance images without actually downloading them.)

We have a sister project called nbdkit (https://github.com/libguestfs/nbdkit). This project lets you turn any disk source into an NBD server. Libguestfs can access NBD servers directly, eg:

```
 guestfish −a nbd://remote
```

nbdkit is liberally licensed, so you can link it to or include it in proprietary libraries and code. It also has a simple, stable plugin API so you can easily write plugins against the API which will continue to work in future.

Error opening VMDK disks: ''uses a vmdk feature which is not supported by this qemu version: VMDK version 3''

Qemu (and hence libguestfs) only supports certain VMDK disk images. Others won't work, giving this or similar errors.

Ideally someone would fix qemu to support the latest VMDK features, but in the meantime you have three options:

1. If the guest is hosted on a live, reachable ESX server, then locate and download the disk image called *somename−flat.vmdk*. Despite the name, this is a raw disk image, and can be opened by anything.

   If you have a recent enough version of qemu and libguestfs, then you may be able to access this disk image remotely using either HTTPS or ssh. See "REMOTE STORAGE" in **guestfs**(3).

2. Use VMware's proprietary vdiskmanager tool to convert the image to raw format.
3. Use nbdkit with the proprietary VDDK plugin to live export the disk image as an NBD source. This should allow you to read and write the VMDK file.

UFS disks (as used by BSD) cannot be opened.

The UFS filesystem format has many variants, and these are not self-identifying. The Linux kernel has to be told which variant of UFS it has to use, which libguestfs cannot know.

You have to pass the right `ufstype` mount option when mounting these filesystems.

See https://www.kernel.org/doc/Documentation/filesystems/ufs.txt

Windows ReFS

Windows ReFS is Microsoft's ZFS/Btrfs copy. This filesystem has not yet been reverse engineered and implemented in the Linux kernel, and therefore libguestfs doesn't support it. At the moment it seems to be very rare "in the wild".

Non-ASCII characters don't appear on VFAT filesystems.

Typical symptoms of this problem:

- You get an error when you create a file where the filename contains non-ASCII characters, particularly non 8−bit characters from Asian languages (Chinese, Japanese, etc). The filesystem is VFAT.
- When you list a directory from a VFAT filesystem, filenames appear as question marks.

This is a design flaw of the GNU/Linux system.

VFAT stores long filenames as UTF−16 characters. When opening or returning filenames, the Linux kernel has to translate these to some form of 8 bit string. UTF−8 would be the obvious choice, except for Linux users who persist in using non−UTF−8 locales (the user's locale is not known to the kernel because it's a function of libc).

Therefore you have to tell the kernel what translation you want done when you mount the filesystem. The two methods are the `iocharset` parameter (which is not relevant to libguestfs) and the `utf8` flag.

So to use a VFAT filesystem you must add the `utf8` flag when mounting. From guestfish, use:

```
><fs> mount−options utf8 /dev/sda1 /
```

or on the guestfish command line:

```
guestfish [...] −m /dev/sda1:/:utf8
```

or from the API:

```
guestfs_mount_options (g, "utf8", "/dev/sda1", "/");
```

The kernel will then translate filenames to and from UTF−8 strings.

We considered adding this mount option transparently, but unfortunately there are several problems with doing that:

- On some Linux systems, the `utf8` mount option doesn't work. We don't precisely understand what systems or why, but this was reliably reported by one user.
- It would prevent you from using the `iocharset` parameter because it is incompatible with `utf8`. It is probably not a good idea to use this parameter, but we don't want to prevent it.

Non-ASCII characters appear as underscore (_) on ISO9660 filesystems.

The filesystem was not prepared correctly with mkisofs or genisoimage. Make sure the filesystem was created using Joliet and/or Rock Ridge extensions. libguestfs does not require any special mount options to handle the filesystem.

Cannot open Windows guests which use NTFS.

You see errors like:

```
mount: unknown filesystem type 'ntfs'
```

On Red Hat Enterprise Linux or CentOS < 7.2, you have to install the libguestfs-winsupport package. In RHEL ≥ 7.2, `libguestfs−winsupport` is part of the base RHEL distribution, but see the next question.

"mount: unsupported filesystem type" with NTFS in RHEL ≥ 7.2

In RHEL 7.2 we were able to add `libguestfs−winsupport` to the base RHEL distribution, but we had to disable the ability to use it for opening and editing filesystems. It is only supported when used with **virt−v2v**(1). If you try to use**guestfish**(1) or **guestmount**(1) or some other programs on an NTFS filesystem, you will see the error:

```
mount: unsupported filesystem type
```

This is not a supported configuration, and it will not be made to work in RHEL. Don't bother to open a bug about it, as it will be immediately `CLOSED −> WONTFIX`.

You may compile your own libguestfs removing this restriction, but that won't be endorsed or supported by Red Hat.

Cannot open or inspect RHEL 7 guests.
Cannot open Linux guests which use XFS.

RHEL 7 guests, and any other guests that use XFS, can be opened by libguestfs, but you have to install the `libguestfs−xfs` package.

USING LIBGUESTFS IN YOUR OWN PROGRAMS

The API has hundreds of methods, where do I start?

We recommend you start by reading the API overview: "API OVERVIEW" in **guestfs**(3).

Although the API overview covers the C API, it is still worth reading even if you are going to use another programming language, because the API is the same, just with simple logical changes to the names of the calls:

```
                  C  guestfs_ln_sf (g, target, linkname);
             Python  g.ln_sf (target, linkname);
              OCaml  g#ln_sf target linkname;
               Perl  $g−>ln_sf (target, linkname);
  Shell (guestfish)  ln−sf target linkname
                PHP  guestfs_ln_sf ($g, $target, $linkname);
```

Once you're familiar with the API overview, you should look at this list of starting points for other language bindings: "USING LIBGUESTFS WITH OTHER PROGRAMMING LANGUAGES" in **guestfs**(3).

Can I use libguestfs in my proprietary / closed source / commercial program?

In general, yes. However this is not legal advice − read the license that comes with libguestfs, and if you have specific questions contact a lawyer.

In the source tree the license is in the file `COPYING.LIB` (LGPLv2+ for the library and bindings) and `COPYING` (GPLv2+ for the standalone programs).

DEBUGGING LIBGUESTFS

Help, it's not working!

If no libguestfs program seems to work at all, run the program below and paste the **complete, unedited** output into an email to `libguestfs` @ `redhat.com`:

```
libguestfs−test−tool
```

If a particular operation fails, supply all the information in this checklist, in an email to `libguestfs` @ `redhat.com`:

1. What are you trying to do?
2. What exact command(s) did you run?
3. What was the precise error or output of these commands?

4. Enable debugging, run the commands again, and capture the **complete** output. **Do not edit the output.**

   ```
   export LIBGUESTFS_DEBUG=1
   export LIBGUESTFS_TRACE=1
   ```

5. Include the version of libguestfs, the operating system version, and how you installed libguestfs (eg. from source, `yum install`, etc.)

How do I debug when using any libguestfs program or tool (eg. virt-customize or virt-df)?

There are two `LIBGUESTFS_*` environment variables you can set in order to get more information from libguestfs.

`LIBGUESTFS_TRACE`
: Set this to 1 and libguestfs will print out each command / API call in a format which is similar to guestfish commands.

`LIBGUESTFS_DEBUG`
: Set this to 1 in order to enable massive amounts of debug messages. If you think there is some problem inside the libguestfs appliance, then you should use this option.

To set these from the shell, do this before running the program:

```
export LIBGUESTFS_TRACE=1
export LIBGUESTFS_DEBUG=1
```

For csh/tcsh the equivalent commands would be:

```
setenv LIBGUESTFS_TRACE 1
setenv LIBGUESTFS_DEBUG 1
```

For further information, see: "ENVIRONMENT VARIABLES" in **guestfs**(3).

How do I debug when using guestfish?

You can use the same environment variables above. Alternatively use the guestfish options −x (to trace commands) or −v (to get the full debug output), or both.

For further information, see: **guestfish**(1).

How do I debug when using the API?

Call "guestfs_set_trace" in **guestfs**(3) to enable command traces, and/or "guestfs_set_verbose" in **guestfs**(3) to enable debug messages.

For best results, call these functions as early as possible, just after creating the guestfs handle if you can, and definitely before calling launch.

How do I capture debug output and put it into my logging system?

Use the event API. For examples, see: "SETTING CALLBACKS TO HANDLE EVENTS" in **guestfs**(3) and the *examples/debug−logging.c* program in the libguestfs sources.

Digging deeper into the appliance boot process.

Enable debugging and then read this documentation on the appliance boot process: **guestfs−internals**(1).

libguestfs hangs or fails during run/launch.

Enable debugging and look at the full output. If you cannot work out what is going on, file a bug report, including the *complete* output of **libguestfs−test−tool**(1).

Debugging libvirt

If you are using the libvirt backend, and libvirt is failing, then you can enable debugging by editing */etc/libvirt/libvirtd.conf*.

If you are running as non-root, then you have to edit a different file. Create *~/.config/libvirt/libvirtd.conf* containing:

```
log_level=1
log_outputs="1:file:/tmp/libvirtd.log"
```

Kill any session (non-root) libvirtd that is running, and next time you run the libguestfs command, you should see a large amount of useful debugging information from libvirtd in */tmp/libvirtd.log*

Broken kernel, or trying a different kernel.

You can choose a different kernel for the appliance by setting some supermin environment variables:

```
export SUPERMIN_KERNEL_VERSION=4.8.0−1.fc25.x86_64
export SUPERMIN_KERNEL=/boot/vmlinuz−$SUPERMIN_KERNEL_VERSION
export SUPERMIN_MODULES=/lib/modules/$SUPERMIN_KERNEL_VERSION
rm −rf /var/tmp/.guestfs−*
libguestfs−test−tool
```

Broken qemu, or trying a different qemu.

You can choose a different qemu by setting the hypervisor environment variable:

```
export LIBGUESTFS_HV=/path/to/qemu−system−x86_64
libguestfs−test−tool
```

DESIGN/INTERNALS OF LIBGUESTFS

See also **guestfs−internals** (1).

Why don't you do everything through the FUSE / filesystem interface?

We offer a command called **guestmount** (1) which lets you mount guest filesystems on the host. This is implemented as a FUSE module. Why don't we just implement the whole of libguestfs using this mechanism, instead of having the large and rather complicated API?

The reasons are twofold. Firstly, libguestfs offers API calls for doing things like creating and deleting partitions and logical volumes, which don't fit into a filesystem model very easily. Or rather, you could fit them in: for example, creating a partition could be mapped to `mkdir /fs/hda1` but then you'd have to specify some method to choose the size of the partition (maybe `echo 100M > /fs/hda1/.size`), and the partition type, start and end sectors etc., but once you've done that the filesystem-based API starts to look more complicated than the call-based API we currently have.

The second reason is for efficiency. FUSE itself is reasonably efficient, but it does make lots of small, independent calls into the FUSE module. In guestmount these have to be translated into messages to the libguestfs appliance which has a big overhead (in time and round trips). For example, reading a file in 64 KB chunks is inefficient because each chunk would turn into a single round trip. In the libguestfs API it is much more efficient to download an entire file or directory through one of the streaming calls like `guestfs_download` or `guestfs_tar_out`.

Why don't you do everything through GVFS?

The problems are similar to the problems with FUSE.

GVFS is a better abstraction than POSIX/FUSE. There is an FTP backend for GVFS, which is encouraging because FTP is conceptually similar to the libguestfs API. However the GVFS FTP backend makes multiple simultaneous connections in order to keep interactivity, which we can't easily do with libguestfs.

Why can I write to the disk, even though I added it read-only?
Why does `−−ro` appear to have no effect?

When you add a disk read-only, libguestfs places a writable overlay on top of the underlying disk. Writes go into this overlay, and are discarded when the handle is closed (or `guestfish` etc. exits).

There are two reasons for doing it this way: Firstly read-only disks aren't possible in many cases (eg. IDE simply doesn't support them, so you couldn't have an IDE-emulated read-only disk, although this is not common in real libguestfs installations).

Secondly and more importantly, even if read-only disks were possible, you wouldn't want them. Mounting any filesystem that has a journal, even `mount −o ro`, causes writes to the filesystem because the journal has to be replayed and metadata updated. If the disk was truly read-only, you wouldn't be able to mount a dirty filesystem.

To make it usable, we create the overlay as a place to temporarily store these writes, and then we discard it afterwards. This ensures that the underlying disk is always untouched.

Note also that there is a regression test for this when building libguestfs (in `tests/qemu`). This is one reason why it's important for packagers to run the test suite.

Does `--ro` make all disks read-only?

No! The `--ro` option only affects disks added on the command line, ie. using `-a` and `-d` options.

In guestfish, if you use the `add` command, then disk is added read-write (unless you specify the `readonly:true` flag explicitly with the command).

Can I use `guestfish --ro` as a way to backup my virtual machines?

Usually this is *not* a good idea. The question is answered in more detail in this mailing list posting: https://www.redhat.com/archives/libguestfs/2010−August/msg00024.html

See also the next question.

Why can't I run fsck on a live filesystem using `guestfish --ro`?

This command will usually *not* work:

```
 guestfish --ro -a /dev/vg/my_root_fs run : fsck /dev/sda
```

The reason for this is that qemu creates a snapshot over the original filesystem, but it doesn't create a strict point-in-time snapshot. Blocks of data on the underlying filesystem are read by qemu at different times as the fsck operation progresses, with host writes in between. The result is that fsck sees massive corruption (imaginary, not real!) and fails.

What you have to do is to create a point-in-time snapshot. If it's a logical volume, use an LVM2 snapshot. If the filesystem is located inside something like a btrfs/ZFS file, use a btrfs/ZFS snapshot, and then run the fsck on the snapshot. In practice you don't need to use libguestfs for this — just run */sbin/fsck* directly.

Creating point-in-time snapshots of host devices and files is outside the scope of libguestfs, although libguestfs can operate on them once they are created.

What's the difference between guestfish and virt-rescue?

A lot of people are confused by the two superficially similar tools we provide:

```
 $ guestfish --ro -a guest.img
 ><fs> run
 ><fs> fsck /dev/sda1

 $ virt-rescue --ro guest.img
 ><rescue> /sbin/fsck /dev/sda1
```

And the related question which then arises is why you can't type in full shell commands with all the −−options in guestfish (but you can in **virt−rescue**(1)).

guestfish(1) is a program providing structured access to the **guestfs**(3) API. It happens to be a nice interactive shell too, but its primary purpose is structured access from shell scripts. Think of it more like a language binding, like Python and other bindings, but for shell. The key differentiating factor of guestfish (and the libguestfs API in general) is the ability to automate changes.

virt−rescue(1) is a free-for-all freeform way to boot the libguestfs appliance and make arbitrary changes to your VM. It's not structured, you can't automate it, but for making quick ad-hoc fixes to your guests, it can be quite useful.

But, libguestfs also has a "backdoor" into the appliance allowing you to send arbitrary shell commands. It's not as flexible as virt-rescue, because you can't interact with the shell commands, but here it is anyway:

```
 ><fs> debug sh "cmd arg1 arg2 ..."
```

Note that you should **not** rely on this. It could be removed or changed in future. If your program needs some operation, please add it to the libguestfs API instead.

What's the deal with `guestfish -i`?
Why does virt-cat only work on a real VM image, but virt-df works on any disk image?
What does "no root device found in this operating system image" mean?

These questions are all related at a fundamental level which may not be immediately obvious.

At the **guestfs**(3) API level, a "disk image" is just a pile of partitions and filesystems.

In contrast, when the virtual machine boots, it mounts those filesystems into a consistent hierarchy such as:

```
 /          (/dev/sda2)
 |
 +-- /boot  (/dev/sda1)
 |
 +-- /home  (/dev/vg_external/Homes)
 |
 +-- /usr   (/dev/vg_os/lv_usr)
 |
 +-- /var   (/dev/vg_os/lv_var)
```

(or drive letters on Windows).

The API first of all sees the disk image at the "pile of filesystems" level. But it also has a way to inspect the disk image to see if it contains an operating system, and how the disks are mounted when the operating system boots: "INSPECTION" in **guestfs**(3).

Users expect some tools (like **virt−cat**(1)) to work with VM paths:

```
 virt-cat fedora.img /var/log/messages
```

How does virt-cat know that */var* is a separate partition? The trick is that virt-cat performs inspection on the disk image, and uses that to translate the path correctly.

Some tools (including **virt−cat**(1), **virt−edit**(1), **virt−ls**(1)) use inspection to map VM paths. Other tools, such as **virt−df**(1) and **virt−filesystems**(1) operate entirely at the raw "big pile of filesystems" level of the libguestfs API, and don't use inspection.

guestfish(1) is in an interesting middle ground. If you use the *−a* and *−m* command line options, then you have to tell guestfish exactly how to add disk images and where to mount partitions. This is the raw API level.

If you use the *−i* option, libguestfs performs inspection and mounts the filesystems for you.

The error `no root device found in this operating system image` is related to this. It means inspection was unable to locate an operating system within the disk image you gave it. You might see this from programs like virt-cat if you try to run them on something which is just a disk image, not a virtual machine disk image.

What do these `debug*` and `internal-*` functions do?

There are some functions which are used for debugging and internal purposes which are *not* part of the stable API.

The `debug*` (or `guestfs_debug*`) functions, primarily "guestfs_debug" in **guestfs**(3) and a handful of others, are used for debugging libguestfs. Although they are not part of the stable API and thus may change or be removed at any time, some programs may want to call these while waiting for features to be added to libguestfs.

The `internal-*` (or `guestfs_internal_*`) functions are purely to be used by libguestfs itself. There is no reason for programs to call them, and programs should not try to use them. Using them will often cause bad things to happen, as well as not being part of the documented stable API.

DEVELOPERS

Where do I send patches?

Please send patches to the libguestfs mailing list https://www.redhat.com/mailman/listinfo/libguestfs. You don't have to be subscribed, but there will be a delay until your posting is manually approved.

Please don't use github pull requests − they will be ignored. The reasons are (a) we want to discuss and dissect patches on the mailing list, and (b) github pull requests turn into merge commits but we prefer to have a linear history.

How do I propose a feature?

Large new features that you intend to contribute should be discussed on the mailing list first (https://www.redhat.com/mailman/listinfo/libguestfs). This avoids disappointment and wasted work if we don't think the feature would fit into the libguestfs project.

If you want to suggest a useful feature but don't want to write the code, you can file a bug (see "GETTING HELP AND REPORTING BUGS") with `"RFE: "` at the beginning of the Summary line.

Who can commit to libguestfs git?

About 5 people have commit access to github. Patches should be posted on the list first and ACKed. The policy for ACKing and pushing patches is outlined here:

https://www.redhat.com/archives/libguestfs/2012−January/msg00023.html

Can I fork libguestfs?

Of course you can. Git makes it easy to fork libguestfs. Github makes it even easier. It's nice if you tell us on the mailing list about forks and the reasons for them.

MISCELLANEOUS QUESTIONS

Can I monitor the live disk activity of a virtual machine using libguestfs?

A common request is to be able to use libguestfs to monitor the live disk activity of a guest, for example, to get notified every time a guest creates a new file. Libguestfs does *not* work in the way some people imagine, as you can see from this diagram:

```
               │ monitoring program using libguestfs │

                                ↓

 │ live VM   │    │ libguestfs appliance │

 │ kernel (1)│    │ appliance kernel (2) │

      ↓                         ↓ (r/o connection)

      |      disk image         |
```

This scenario is safe (as long as you set the `readonly` flag when adding the drive). However the libguestfs appliance kernel (2) does not see all the changes made to the disk image, for two reasons:

i. The VM kernel (1) can cache data in memory, so it doesn't appear in the disk image.

ii. The libguestfs appliance kernel (2) doesn't expect that the disk image is changing underneath it, so its own cache is not magically updated even when the VM kernel (1) does update the disk image.

The only supported solution is to restart the entire libguestfs appliance whenever you want to look at changes in the disk image. At the API level that corresponds to calling `guestfs_shutdown` followed by `guestfs_launch`, which is a heavyweight operation (see also **guestfs−performance**(3)).

There are some unsupported hacks you can try if relaunching the appliance is really too costly:

- Call `guestfs_drop_caches (g, 3)`. This causes all cached data help by the libguestfs appliance kernel (2) to be discarded, so it goes back to the disk image.

  However this on its own is not sufficient, because qemu also caches some data. You will also need to patch libguestfs to (re−)enable the `cache=none` mode. See: https://rwmj.wordpress.com/2013/09/02/new−in−libguestfs−allow−cache−mode−to−be−selected/

- Use a tool like virt-bmap instead.
- Run an agent inside the guest.

Nothing helps if the guest is making more fundamental changes (eg. deleting filesystems). For those kinds of things you must relaunch the appliance.

(Note there is a third problem that you need to use consistent snapshots to really examine live disk images, but that's a general problem with using libguestfs against any live disk image.)

SEE ALSO

guestfish (1), **guestfs** (3), http://libguestfs.org/.

AUTHORS

Richard W.M. Jones (`rjones at redhat dot com`)

COPYRIGHT

LICENSE

BUGS

To get a list of bugs against libguestfs, use this link: https://bugzilla.redhat.com/buglist.cgi?component=libguestfs&product=Virtualization+Tools

To report a new bug against libguestfs, use this link: https://bugzilla.redhat.com/enter_bug.cgi?component=libguestfs&product=Virtualization+Tools

When reporting a bug, please supply:

- The version of libguestfs.
- Where you got libguestfs (eg. which Linux distro, compiled from source, etc)
- Describe the bug accurately and give a way to reproduce it.
- Run **libguestfs−test−tool** (1) and paste the **complete, unedited** output into the bug report.