

**NAME**

guestfish – the guest filesystem shell

**SYNOPSIS**

```
guestfish [--options] [commands]

guestfish

guestfish [--ro|--rw] -a disk.img

guestfish [--ro|--rw] -a disk.img -m dev[:mountpoint]

guestfish -d libvirt-domain

guestfish [--ro|--rw] -a disk.img -i

guestfish -d libvirt-domain -i
```

**WARNING**

Using guestfish in write mode on live virtual machines, or concurrently with other disk editing tools, can be dangerous, potentially causing disk corruption. The virtual machine must be shut down before you use this command, and disk images must not be edited concurrently.

Use the `--ro` (read-only) option to use guestfish safely if the disk image or virtual machine might be live. You may see strange or inconsistent results if running concurrently with other changes, but with this option you won't risk disk corruption.

**DESCRIPTION**

Guestfish is a shell and command-line tool for examining and modifying virtual machine filesystems. It uses libguestfs and exposes all of the functionality of the guestfs API, see **guestfs**(3).

Guestfish gives you structured access to the libguestfs API, from shell scripts or the command line or interactively. If you want to rescue a broken virtual machine image, you should look at the **virt-rescue**(1) command.

**EXAMPLES****As an interactive shell**

```
$ guestfish

Welcome to guestfish, the guest filesystem shell for
editing virtual machine filesystems.

Type: 'help' for a list of commands
      'man' to read the manual
      'quit' to quit the shell

><fs> add-ro disk.img
><fs> run
><fs> list-filesystems
/dev/sda1: ext4
/dev/vg_guest/lv_root: ext4
/dev/vg_guest/lv_swap: swap
><fs> mount /dev/vg_guest/lv_root /
><fs> cat /etc/fstab
# /etc/fstab
# Created by anaconda
[...]
><fs> exit
```

**From shell scripts**

Create a new */etc/motd* file in a guest or disk image:

```
guestfish <<_EOF_
add disk.img
run
mount /dev/vg_guest/lv_root /
write /etc/motd "Welcome, new users"
_EOF_
```

List the LVM logical volumes in a disk image:

```
guestfish -a disk.img --ro <<_EOF_
run
lvs
_EOF_
```

List all the filesystems in a disk image:

```
guestfish -a disk.img --ro <<_EOF_
run
list-fileSYSTEMS
_EOF_
```

**On one command line**

Update */etc/resolv.conf* in a guest:

```
guestfish \
  add disk.img : run : mount /dev/vg_guest/lv_root / : \
  write /etc/resolv.conf "nameserver 1.2.3.4"
```

Edit */boot/grub/grub.conf* interactively:

```
guestfish --rw --add disk.img \
  --mount /dev/vg_guest/lv_root \
  --mount /dev/sda1:/boot \
  edit /boot/grub/grub.conf
```

**Mount disks automatically**

Use the *-i* option to automatically mount the disks from a virtual machine:

```
guestfish --ro -a disk.img -i cat /etc/group

guestfish --ro -d libvirt-domain -i cat /etc/group
```

Another way to edit */boot/grub/grub.conf* interactively is:

```
guestfish --rw -a disk.img -i edit /boot/grub/grub.conf
```

**As a script interpreter**

Create a 100MB disk containing an ext2-formatted partition:

```
#!/usr/bin/guestfish -f
sparse test1.img 100M
run
part-disk /dev/sda mbr
mkfs ext2 /dev/sda1
```

**Start with a prepared disk**

Create a 1G disk called *test1.img* containing a single ext2-formatted partition:

```
guestfish -N fs
```

To list what is available do:

```
guestfish -N help | less
```

### Remote drives

Access a remote disk using ssh:

```
guestfish -a ssh://example.com/path/to/disk.img
```

### Remote control

```
eval "`guestfish --listen`"
guestfish --remote add-ro disk.img
guestfish --remote run
guestfish --remote lvs
```

## OPTIONS

### --help

Displays general help on options.

### -h

### --cmd-help

Lists all available guestfish commands.

### -h CMD

### --cmd-help CMD

Displays detailed help on a single command `cmd`.

### -a IMAGE

### --add IMAGE

Add a block device or virtual machine image to the shell.

The format of the disk image is auto-detected. To override this and force a particular format use the `--format=..` option.

Using this flag is mostly equivalent to using the `add` command, with `readonly:true` if the `--ro` flag was given, and with `format:...` if the `--format=...` flag was given.

### -a URI

### --add URI

Add a remote disk. See “ADDING REMOTE STORAGE”.

### --blocksize=512

### --blocksize=4096

### --blocksize

This parameter sets the sector size of the disk image. It affects all explicitly added subsequent disks after this parameter. Using `--blocksize` with no argument switches the disk sector size to the default value which is usually 512 bytes. See also “`guestfs_add_drive_opts`” in `guestfs(3)`.

### -c URI

### --connect URI

When used in conjunction with the `-d` option, this specifies the libvirt URI to use. The default is to use the default libvirt connection.

### --csh

If using the `--listen` option and a csh-like shell, use this option. See section “REMOTE CONTROL AND CSH” below.

### -d LIBVIRT-DOMAIN

### --domain LIBVIRT-DOMAIN

Add disks from the named libvirt domain. If the `--ro` option is also used, then any libvirt domain can be used. However in write mode, only libvirt domains which are shut down can be named here.

Domain UUIDs can be used instead of names.

Using this flag is mostly equivalent to using the `add-domain` command, with `readonly:true` if the `--ro` flag was given, and with `format:...` if the `--format=...` flag was given.

**--echo-keys**

When prompting for keys and passphrases, guestfish normally turns echoing off so you cannot see what you are typing. If you are not worried about Tempest attacks and there is no one else in the room you can specify this flag to see what you are typing.

**-f FILE****--file FILE**

Read commands from FILE. To write pure guestfish scripts, use:

```
#!/usr/bin/guestfish -f
```

**--format=raw|qcow2|..****--format**

The default for the *-a* option is to auto-detect the format of the disk image. Using this forces the disk format for *-a* options which follow on the command line. Using *--format* with no argument switches back to auto-detection for subsequent *-a* options.

For example:

```
guestfish --format=raw -a disk.img
```

forces raw format (no auto-detection) for *disk.img*.

```
guestfish --format=raw -a disk.img --format -a another.img
```

forces raw format (no auto-detection) for *disk.img* and reverts to auto-detection for *another.img*.

If you have untrusted raw-format guest disk images, you should use this option to specify the disk format. This avoids a possible security problem with malicious guests (CVE-2010-3851). See also “add”.

**-i****--inspector**

Using **virt-inspector**(1) code, inspect the disks looking for an operating system and mount filesystems as they would be mounted on the real virtual machine.

Typical usage is either:

```
guestfish -d myguest -i
```

(for an inactive libvirt domain called *myguest*), or:

```
guestfish --ro -d myguest -i
```

(for active domains, readonly), or specify the block device directly:

```
guestfish --rw -a /dev/Guests/MyGuest -i
```

Note that the command line syntax changed slightly over older versions of guestfish. You can still use the old syntax:

```
guestfish [--ro] -i disk.img
```

```
guestfish [--ro] -i libvirt-domain
```

Using this flag is mostly equivalent to using the *inspect-os* command and then using other commands to mount the filesystems that were found.

**--key SELECTOR**

Specify a key for LUKS, to automatically open a LUKS device when using the inspection. ID can be either the libguestfs device name, or the UUID of the LUKS device.

**--key ID:key:KEY\_STRING**

Use the specified KEY\_STRING as passphrase.

**--key** ID:file:FILENAME

Read the passphrase from *FILENAME*.

**--keys-from-stdin**

Read key or passphrase parameters from stdin. The default is to try to read passphrases from the user by opening */dev/tty*.

If there are multiple encrypted devices then you may need to supply multiple keys on stdin, one per line.

**--listen**

Fork into the background and listen for remote commands. See section “REMOTE CONTROL GUESTFISH OVER A SOCKET” below.

**--live**

Connect to a live virtual machine. (Experimental, see “ATTACHING TO RUNNING DAEMONS” in **guestfs(3)**).

**-m** dev[:mountpoint[:options[:fstype]]]

**--mount** dev[:mountpoint[:options[:fstype]]]

Mount the named partition or logical volume on the given mountpoint.

If the mountpoint is omitted, it defaults to */*.

You have to mount something on */* before most commands will work.

If any *-m* or *--mount* options are given, the guest is automatically launched.

If you don’t know what filesystems a disk image contains, you can either run *guestfish* without this option, then list the partitions, filesystems and LVs available (see “list-partitions”, “list-filesystems” and “lvs” commands), or you can use the **virt-filesystems(1)** program.

The third (and rarely used) part of the mount parameter is the list of mount options used to mount the underlying filesystem. If this is not given, then the mount options are either the empty string or *ro* (the latter if the *--ro* flag is used). By specifying the mount options, you override this default choice. Probably the only time you would use this is to enable ACLs and/or extended attributes if the filesystem can support them:

```
-m /dev/sda1::acl,user_xattr
```

Using this flag is equivalent to using the *mount-options* command.

The fourth part of the parameter is the filesystem driver to use, such as *ext3* or *ntfs*. This is rarely needed, but can be useful if multiple drivers are valid for a filesystem (eg: *ext2* and *ext3*), or if *libguestfs* misidentifies a filesystem.

**--network**

Enable QEMU user networking in the guest.

**-N** [FILENAME=]TYPE

**--new** [FILENAME=]TYPE

**-N help**

Prepare a fresh disk image formatted as *TYPE*. This is an alternative to the *-a* option: whereas *-a* adds an existing disk, *-N* creates a preformatted disk with a filesystem and adds it. See “PREPARED DISK IMAGES” below.

**-n**

**--no-sync**

Disable autosync. This is enabled by default. See the discussion of autosync in the **guestfs(3)** manpage.

**--no-dest-paths**

Don’t tab-complete paths on the guest filesystem. It is useful to be able to hit the tab key to complete paths on the guest filesystem, but this causes extra “hidden” *guestfs* calls to be made, so this option is

here to allow this feature to be disabled.

**--pipe-error**

If writes fail to pipe commands (see “PIPES” below), then the command returns an error.

The default (also for historical reasons) is to ignore such errors so that:

```
><fs> command_with_lots_of_output | head
```

doesn't give an error.

**--progress-bars**

Enable progress bars, even when guestfish is used non-interactively.

Progress bars are enabled by default when guestfish is used as an interactive shell.

**--no-progress-bars**

Disable progress bars.

**--remote**

**--remote=PID**

Send remote commands to \$GUESTFISH\_PID or pid. See section “REMOTE CONTROL GUESTFISH OVER A SOCKET” below.

**-r**

**--ro**

This changes the *-a*, *-d* and *-m* options so that disks are added and mounts are done read-only.

The option must always be used if the disk image or virtual machine might be running, and is generally recommended in cases where you don't need write access to the disk.

Note that prepared disk images created with *-N* are not affected by this option. Also commands like *add* are not affected – you have to specify the `readonly:true` option explicitly if you need it.

See also “OPENING DISKS FOR READ AND WRITE” below.

**--selinux**

This option is provided for backwards compatibility and does nothing.

**-v**

**--verbose**

Enable very verbose messages. This is particularly useful if you find a bug.

**-V**

**--version**

Display the guestfish / libguestfs version number and exit.

**-w**

**--rw**

This changes the *-a*, *-d* and *-m* options so that disks are added and mounts are done read-write.

See “OPENING DISKS FOR READ AND WRITE” below.

**-x** Echo each command before executing it.

## COMMANDS ON COMMAND LINE

Any additional (non-option) arguments are treated as commands to execute.

Commands to execute should be separated by a colon (:), where the colon is a separate parameter. Thus:

```
guestfish cmd [args...] : cmd [args...] : cmd [args...] ...
```

If there are no additional arguments, then we enter a shell, either an interactive shell with a prompt (if the input is a terminal) or a non-interactive shell.

In either command line mode or non-interactive shell, the first command that gives an error causes the whole shell to exit. In interactive mode (with a prompt) if a command fails, you can continue to enter

commands.

Note that arguments of the commands will be considered as guestfish options if they start with a dash (-): you can always separate the guestfish options and the rest of the commands (with their arguments) using a double dash (--). For example:

```
guestfish -- disk_create overlay.qcow2 qcow2 -l backingfile:image.img
```

## USING launch (OR run)

As with **guestfs**(3), you must first configure your guest by adding disks, then launch it, then mount any disks you need, and finally issue actions/commands. So the general order of the day is:

- add or `-a/--add`
- launch (aka run)
- mount or `-m/--mount`
- any other commands

`run` is a synonym for `launch`. You must launch (or run) your guest before mounting or performing any other commands.

The only exception is that if any of the `-i`, `-m`, `--mount`, `-N` or `--new` options were given then `run` is done automatically, simply because guestfish can't perform the action you asked for without doing this.

## OPENING DISKS FOR READ AND WRITE

The guestfish, **guestmount**(1) and **virt-rescue**(1) options `--ro` and `--rw` affect whether the other command line options `-a`, `-c`, `-d`, `-i` and `-m` open disk images read-only or for writing.

In `libguestfs` ≤ 1.10, guestfish, guestmount and virt-rescue defaulted to opening disk images supplied on the command line for write. To open a disk image read-only you have to do `-a image --ro`.

This matters: If you accidentally open a live VM disk image writable then you will cause irreversible disk corruption.

In a future `libguestfs` we intend to change the default the other way. Disk images will be opened read-only. You will have to either specify `guestfish --rw`, `guestmount --rw`, `virt-rescue --rw`, or change the configuration file in order to get write access for disk images specified by those other command line options.

This version of guestfish, guestmount and virt-rescue has a `--rw` option which does nothing (it is already the default). However it is highly recommended that you use this option to indicate that you need write access, and prepare your scripts for the day when this option will be required for write access.

**Note:** This does *not* affect commands like “add” and “mount”, or any other `libguestfs` program apart from guestfish and guestmount.

## QUOTING

You can quote ordinary parameters using either single or double quotes. For example:

```
add "file with a space.img"
```

```
rm '/file name'
```

```
rm '/'
```

A few commands require a list of strings to be passed. For these, use a whitespace-separated list, enclosed in quotes. Strings containing whitespace to be passed through must be enclosed in single quotes. A literal single quote must be escaped with a backslash.

```
vgcreate VG "/dev/sda1 /dev/sdb1"
command "/bin/echo 'foo      bar'"
command "/bin/echo \"'foo\""
```

## ESCAPE SEQUENCES IN DOUBLE QUOTED ARGUMENTS

In double-quoted arguments (only) use backslash to insert special characters:

`\a` Alert (bell) character.

`\b` Backspace character.

`\f` Form feed character.

`\n` Newline character.

`\r` Carriage return character.

`\t` Horizontal tab character.

`\v` Vertical tab character.

`\"` A literal double quote character.

`\ooo`

A character with octal value *ooo*. There must be precisely 3 octal digits (unlike C).

`\xhh`

A character with hex value *hh*. There must be precisely 2 hex digits.

In the current implementation `\000` and `\x00` cannot be used in strings.

`\\` A literal backslash character.

## OPTIONAL ARGUMENTS

Some commands take optional arguments. These arguments appear in this documentation as `[argname:..]`. You can use them as in these examples:

```
add filename
```

```
add filename readonly:true
```

```
add filename format:qcow2 readonly:false
```

Each optional argument can appear at most once. All optional arguments must appear after the required ones.

## NUMBERS

This section applies to all commands which can take integers as parameters.

### SIZE SUFFIX

When the command takes a parameter measured in bytes, you can use one of the following suffixes to specify kilobytes, megabytes and larger sizes:

**k** or **K** or **KiB**

The size in kilobytes (multiplied by 1024).

**KB** The size in SI 1000 byte units.

**M** or **MiB**

The size in megabytes (multiplied by 1048576).

**MB** The size in SI 1000000 byte units.

**G** or **GiB**

The size in gigabytes (multiplied by 2\*\*30).

**GB** The size in SI 10\*\*9 byte units.

**T** or **TiB**

The size in terabytes (multiplied by 2\*\*40).

**TB** The size in SI 10\*\*12 byte units.



**P or PiB**

The size in petabytes (multiplied by  $2^{50}$ ).

**PB** The size in SI  $10^{15}$  byte units.

**E or EiB**

The size in exabytes (multiplied by  $2^{60}$ ).

**EB** The size in SI  $10^{18}$  byte units.

**Z or ZiB**

The size in zettabytes (multiplied by  $2^{70}$ ).

**ZB** The size in SI  $10^{21}$  byte units.

**Y or YiB**

The size in yottabytes (multiplied by  $2^{80}$ ).

**YB** The size in SI  $10^{24}$  byte units.

For example:

```
truncate-size /file 1G
```

would truncate the file to 1 gigabyte.

Be careful because a few commands take sizes in kilobytes or megabytes (eg. the parameter to “memsize” is specified in megabytes already). Adding a suffix will probably not do what you expect.

**OCTAL AND HEXADECIMAL NUMBERS**

For specifying the radix (base) use the C convention: 0 to prefix an octal number or 0x to prefix a hexadecimal number. For example:

```
1234      decimal number 1234
02322     octal number, equivalent to decimal 1234
0x4d2     hexadecimal number, equivalent to decimal 1234
```

When using the `chmod` command, you almost always want to specify an octal number for the mode, and you must prefix it with 0 (unlike the Unix **chmod**(1) program):

```
chmod 0777 /public # OK
chmod 777 /public  # WRONG! This is mode 777 decimal = 01411 octal.
```

Commands that return numbers usually print them in decimal, but some commands print numbers in other radices (eg. `umask` prints the mode in octal, preceded by 0).

**WILDCARDS AND GLOBBING**

Neither guestfish nor the underlying guestfs API performs wildcard expansion (globbing) by default. So for example the following will not do what you expect:

```
rm-rf /home/*
```

Assuming you don’t have a directory called literally `/home/*` then the above command will return an error.

To perform wildcard expansion, use the `glob` command.

```
glob rm-rf /home/*
```

runs `rm-rf` on each path that matches (ie. potentially running the command many times), equivalent to:

```
rm-rf /home/jim
rm-rf /home/joe
rm-rf /home/mary
```

`glob` only works on simple guest paths and not on device names.

If you have several parameters, each containing a wildcard, then `glob` will perform a Cartesian product.

## COMMENTS

Any line which starts with a `#` character is treated as a comment and ignored. The `#` can optionally be preceded by whitespace, but **not** by a command. For example:

```
# this is a comment
    # this is a comment
foo # NOT a comment
```

Blank lines are also ignored.

## RUNNING COMMANDS LOCALLY

Any line which starts with a `!` character is treated as a command sent to the local shell (*/bin/sh* or whatever **system**(3) uses). For example:

```
!mkdir local
tgz-out /remote local/remote-data.tar.gz
```

will create a directory `local` on the host, and then export the contents of `/remote` on the mounted filesystem to `local/remote-data.tar.gz`. (See `tgz-out`).

To change the local directory, use the `lcd` command. `!cd` will have no effect, due to the way that subprocesses work in Unix.

## LOCAL COMMANDS WITH INLINE EXECUTION

If a line starts with `<!` then the shell command is executed (as for `!`), but subsequently any output (stdout) of the shell command is parsed and executed as guestfish commands.

Thus you can use shell script to construct arbitrary guestfish commands which are then parsed by guestfish.

For example it is tedious to create a sequence of files (eg. */foo.1* through */foo.100*) using guestfish commands alone. However this is simple if we use a shell script to create the guestfish commands for us:

```
<! for n in `seq 1 100`; do echo write /foo.$n $n; done
```

or with names like */foo.001*:

```
<! for n in `seq 1 100`; do printf "write /foo.%03d %d\n" $n $n; done
```

When using guestfish interactively it can be helpful to just run the shell script first (ie. remove the initial `<` character so it is just an ordinary `!` local command), see what guestfish commands it would run, and when you are happy with those prepend the `<` character to run the guestfish commands for real.

## PIPES

Use command `<space> | command` to pipe the output of the first command (a guestfish command) to the second command (any host command). For example:

```
cat /etc/passwd | awk -F: '$3 == 0 { print }'
```

(where `cat` is the guestfish `cat` command, but `awk` is the host `awk` program). The above command would list all accounts in the guest filesystem which have UID 0, ie. root accounts including backdoors. Other examples:

```
hexdump /bin/ls | head
list-devices | tail -1
tgz-out / - | tar ztf -
```

The space before the pipe symbol is required, any space after the pipe symbol is optional. Everything after the pipe symbol is just passed straight to the host shell, so it can contain redirections, globs and anything else that makes sense on the host side.

To use a literal argument which begins with a pipe symbol, you have to quote it, eg:

```
echo "|"
```

## HOME DIRECTORIES

If a parameter starts with the character `~` then the tilde may be expanded as a home directory path (either `~` for the current user's home directory, or `~user` for another user).

Note that home directory expansion happens for users known *on the host*, not in the guest filesystem.

To use a literal argument which begins with a tilde, you have to quote it, eg:

```
echo "~"
```

## ENCRYPTED DISKS

Libguestfs has some support for Linux guests encrypted according to the Linux Unified Key Setup (LUKS) standard, which includes nearly all whole disk encryption systems used by modern Linux guests, and Windows BitLocker.

Identify encrypted block devices and partitions using “`vfs-type`”:

```
><fs> vfs-type /dev/sda2
crypto_LUKS
```

or:

```
><fs> vfs-type /dev/sda2
BitLocker
```

Then open those devices using “`cryptsetup-open`”. This creates a device-mapper device called `/dev/mapper/name`.

```
><fs> cryptsetup-open /dev/sda2 name
Enter key or passphrase ("key"): <enter the passphrase>
```

For Linux guests you have to tell LVM to scan for volume groups on the newly created mapper device:

```
vgscan
vg-activate-all true
```

The filesystems or logical volumes can now be mounted in the usual way.

Before closing an encrypted device you must unmount any logical volumes on it and deactivate the volume groups by calling `vg-activate false VG` on each one. Then you can close the mapper device:

```
vg-activate false /dev/VG
cryptsetup-close /dev/mapper/name
```

## WINDOWS PATHS

If a path is prefixed with `win:` then you can use Windows-style drive letters and paths (with some limitations). The following commands are equivalent:

```
file /WINDOWS/system32/config/system.LOG

file win:\windows\system32\config\system.log

file WIN:C:\Windows\SYSTEM32\CONFIG\SYSTEM.LOG
```

The parameter is rewritten “behind the scenes” by looking up the position where the drive is mounted, prepending that to the path, changing all backslash characters to forward slash, then resolving the result using “`case-sensitive-path`”. For example if the E: drive was mounted on `/e` then the parameter might be rewritten like this:

```
win:e:\foo\bar => /e/F00/bar
```

This only works in argument positions that expect a path.

## UPLOADING AND DOWNLOADING FILES

For commands such as `upload`, `download`, `tar-in`, `tar-out` and others which upload from or download to a local file, you can use the special filename `-` to mean “from stdin” or “to stdout”. For example:

```
upload - /foo
```

reads stdin and creates from that a file `/foo` in the disk image, and:

```
tar-out /etc - | tar tf -
```

writes the tarball to stdout and then pipes that into the external “tar” command (see “PIPES”).

When using `-` to read from stdin, the input is read up to the end of stdin. You can also use a special “heredoc”-like syntax to read up to some arbitrary end marker:

```
upload -<<END /foo
input line 1
input line 2
input line 3
END
```

Any string of characters can be used instead of `END`. The end marker must appear on a line of its own, without any preceding or following characters (not even spaces).

Note that the `-<<` syntax only applies to parameters used to upload local files (so-called “FileIn” parameters in the generator).

## EXIT ON ERROR BEHAVIOUR

By default, guestfish will ignore any errors when in interactive mode (ie. taking commands from a human over a tty), and will exit on the first error in non-interactive mode (scripts, commands given on the command line).

If you prefix a command with a `-` character, then that command will not cause guestfish to exit, even if that (one) command returns an error.

## REMOTE CONTROL GUESTFISH OVER A SOCKET

Guestfish can be remote-controlled over a socket. This is useful particularly in shell scripts where you want to make several different changes to a filesystem, but you don’t want the overhead of starting up a guestfish process each time.

Start a guestfish server process using:

```
eval "`guestfish --listen`"
```

and then send it commands by doing:

```
guestfish --remote cmd [...]
```

To cause the server to exit, send it the exit command:

```
guestfish --remote exit
```

Note that the server will normally exit if there is an error in a command. You can change this in the usual way. See section “EXIT ON ERROR BEHAVIOUR”.

## CONTROLLING MULTIPLE GUESTFISH PROCESSES

The `eval` statement sets the environment variable `$GUESTFISH_PID`, which is how the `--remote` option knows where to send the commands. You can have several guestfish listener processes running using:

```
eval "`guestfish --listen`"
pid1=$GUESTFISH_PID
eval "`guestfish --listen`"
pid2=$GUESTFISH_PID
...
guestfish --remote=$pid1 cmd
guestfish --remote=$pid2 cmd
```

## REMOTE CONTROL AND CSH

When using csh-like shells (csh, tcsh etc) you have to add the `--csh` option:

```
eval "`guestfish --listen --csh`"
```

**REMOTE CONTROL DETAILS**

Remote control happens over a Unix domain socket called `/tmp/.guestfish- $\$UID$ /socket- $\$PID$` , where  $\$UID$  is the effective user ID of the process, and  $\$PID$  is the process ID of the server.

Guestfish client and server versions must match exactly.

Older versions of guestfish were vulnerable to CVE-2013-4419 (see “CVE-2013-4419” in **guestfs**(3)). This is fixed in the current version.

**USING REMOTE CONTROL ROBUSTLY FROM SHELL SCRIPTS**

From Bash, you can use the following code which creates a guestfish instance, correctly quotes the command line, handles failure to start, and cleans up guestfish when the script exits:

```
#!/bin/bash -

set -e

guestfish[0]="guestfish"
guestfish[1]="--listen"
guestfish[2]="--ro"
guestfish[3]="-a"
guestfish[4]="disk.img"

GUESTFISH_PID=
eval "${guestfish[@]}"
if [ -z "$GUESTFISH_PID" ]; then
    echo "error: guestfish didn't start up, see error messages above"
    exit 1
fi

cleanup_guestfish ()
{
    guestfish --remote -- exit >/dev/null 2>&1 ||:
}
trap cleanup_guestfish EXIT ERR

guestfish --remote -- run

# ...
```

**REMOTE CONTROL DOES NOT WORK WITH `-a` ETC. OPTIONS**

Options such as `-a`, `--add`, `-N`, `--new` etc don't interact properly with remote support. They are processed locally, and not sent through to the remote guestfish. In particular this won't do what you expect:

```
guestfish --remote --add disk.img
```

Don't use these options. Use the equivalent commands instead, eg:

```
guestfish --remote add-drive disk.img
```

or:

```
guestfish --remote
><fs> add disk.img
```

**REMOTE CONTROL RUN COMMAND HANGING**

Using the `run` (or `launch`) command remotely in a command substitution context hangs, ie. don't do (note the backquotes):

```
a=`guestfish --remote run`
```

Since the `run` command produces no output on stdout, this is not useful anyway. For further information

see [https://bugzilla.redhat.com/show\\_bug.cgi?id=592910](https://bugzilla.redhat.com/show_bug.cgi?id=592910).

## PREPARED DISK IMAGES

Use the `-N [filename=]type` or `--new [filename=]type` parameter to select one of a set of preformatted disk images that guestfish can make for you to save typing. This is particularly useful for testing purposes. This option is used instead of the `-a` option, and like `-a` can appear multiple times (and can be mixed with `-a`).

The new disk is called *test1.img* for the first `-N`, *test2.img* for the second and so on. Existing files in the current directory are *overwritten*. You can use a different filename by specifying `filename=` before the type (see examples below).

The type briefly describes how the disk should be sized, partitioned, how filesystem(s) should be created, and how content should be added. Optionally the type can be followed by extra parameters, separated by `:` (colon) characters. For example, `-N fs` creates a default 1G, sparsely-allocated disk, containing a single partition, with the partition formatted as ext2. `-N fs:ext4:2G` is the same, but for an ext4 filesystem on a 2GB disk instead.

Note that the prepared filesystem is not mounted. You would usually have to use the `mount /dev/sda1 /` command or add the `-m /dev/sda1` option.

If any `-N` or `--new` options are given, the libguestfs appliance is automatically launched.

## EXAMPLES

Create a 1G disk with an ext4-formatted partition, called *test1.img* in the current directory:

```
guestfish -N fs:ext4
```

Create a 32MB disk with a VFAT-formatted partition, and mount it:

```
guestfish -N fs:vfat:32M -m /dev/sda1
```

Create a blank 200MB disk:

```
guestfish -N disk:200M
```

Create a blank 200MB disk called *blankdisk.img* (instead of *test1.img*):

```
guestfish -N blankdisk.img=disk:200M
```

### -N disk – create a blank disk

```
guestfish -N [filename=]disk[:size]
```

Create a blank disk, size 1G (by default).

The default size can be changed by supplying an optional parameter.

The optional parameters are:

Name	Default value	
size	1G	the size of the disk image

### -N part – create a partitioned disk

```
guestfish -N [filename=]part[:size[:partition]]
```

Create a disk with a single partition. By default the size of the disk is 1G (the available space in the partition will be a tiny bit smaller) and the partition table will be MBR (old DOS-style).

These defaults can be changed by supplying optional parameters.

The optional parameters are:

Name	Default value	
size	1G	the size of the disk image
partition	mbr	partition table type

### -N fs – create a filesystem

```
guestfish -N [filename=]fs[:filesystem[:size[:partition]]]
```

Create a disk with a single partition, with the partition containing an empty filesystem. This defaults to creating a 1G disk (the available space in the filesystem will be a tiny bit smaller) with an MBR (old DOS-style) partition table and an ext2 filesystem.

These defaults can be changed by supplying optional parameters.

The optional parameters are:

Name	Default value	
filesystem	ext2	the type of filesystem to use
size	1G	the size of the disk image
partition	mbr	partition table type

#### **-N lv – create a disk with logical volume**

```
guestfish -N [filename=]lv[:name[:size[:partition]]]
```

Create a disk with a single partition, set up the partition as an LVM2 physical volume, and place a volume group and logical volume on there. This defaults to creating a 1G disk with the VG and LV called /dev/VG/LV. You can change the name of the VG and LV by supplying an alternate name as the first optional parameter.

Note this does not create a filesystem. Use 'lvfs' to do that.

The optional parameters are:

Name	Default value	
name	/dev/VG/LV	the name of the VG and LV to use
size	1G	the size of the disk image
partition	mbr	partition table type

#### **-N lvfs – create a disk with logical volume and filesystem**

```
guestfish -N [filename=]lvfs[:name[:filesystem[:size[:partition]]]]
```

Create a disk with a single partition, set up the partition as an LVM2 physical volume, and place a volume group and logical volume on there. Then format the LV with a filesystem. This defaults to creating a 1G disk with the VG and LV called /dev/VG/LV, with an ext2 filesystem.

The optional parameters are:

Name	Default value	
name	/dev/VG/LV	the name of the VG and LV to use
filesystem	ext2	the type of filesystem to use
size	1G	the size of the disk image
partition	mbr	partition table type

#### **-N bootroot – create a boot and root filesystem**

```
guestfish -N [filename=]bootroot[:bootfs[:rootfs[:size[:bootsize[:partition]]]]]
```

Create a disk with two partitions, for boot and root filesystem. Format the two filesystems independently. There are several optional parameters which control the exact layout and filesystem types.

The optional parameters are:

Name	Default value	
bootfs	ext2	the type of filesystem to use for boot
rootfs	ext2	the type of filesystem to use for root
size	1G	the size of the disk image
bootsize	128M	the size of the boot filesystem
partition	mbr	partition table type

#### **-N bootrootlv – create a boot and root filesystem using LVM**

```
guestfish -N [filename=]bootrootlv[:name[:bootfs[:rootfs[:size[:bootsize[:partition]]]]]]
```

This is the same as `bootroot` but the root filesystem (only) is placed on a logical volume, named by default `/dev/VG/LV`. There are several optional parameters which control the exact layout.

The optional parameters are:

Name	Default value	
<code>name</code>	<code>/dev/VG/LV</code>	the name of the VG and LV for root
<code>bootfs</code>	<code>ext2</code>	the type of filesystem to use for boot
<code>rootfs</code>	<code>ext2</code>	the type of filesystem to use for root
<code>size</code>	<code>1G</code>	the size of the disk image
<code>bootsize</code>	<code>128M</code>	the size of the boot filesystem
<code>partition</code>	<code>mbr</code>	partition table type

## ADDING REMOTE STORAGE

For API-level documentation on this topic, see “`guestfs_add_drive_opts`” in **guestfs(3)** and “`REMOTE STORAGE`” in **guestfs(3)**.

On the command line, you can use the `-a` option to add network block devices using a URI-style format, for example:

```
guestfish -a ssh://root@example.com/disk.img
```

URIs *cannot* be used with the “add” command. The equivalent command using the API directly is:

```
><fs> add /disk.img protocol:ssh server:tcp:example.com username:root
```

The possible `-a` URI formats are described below.

### **-a disk.img**

#### **-a file:///path/to/disk.img**

Add the local disk image (or device) called *disk.img*.

#### **-a ftp://[user@]example.com[:port]/disk.img**

#### **-a ftps://[user@]example.com[:port]/disk.img**

#### **-a http://[user@]example.com[:port]/disk.img**

#### **-a https://[user@]example.com[:port]/disk.img**

#### **-a tftp://[user@]example.com[:port]/disk.img**

Add a disk located on a remote FTP, HTTP or TFTP server.

The equivalent API command would be:

```
><fs> add /disk.img protocol:(ftp|...) server:tcp:example.com
```

#### **-a gluster://example.com[:port]/volname/image**

Add a disk image located on GlusterFS storage.

The server is the one running `glusterd`, and may be `localhost`.

The equivalent API command would be:

```
><fs> add volname/image protocol:gluster server:tcp:example.com
```

#### **-a iscsi://example.com[:port]/target-iqn-name[/lun]**

Add a disk located on an iSCSI server.

The equivalent API command would be:

```
><fs> add target-iqn-name/lun protocol:iscsi server:tcp:example.com
```

#### **-a nbd://example.com[:port]**

#### **-a nbd://example.com[:port]/exportname**

#### **-a nbd://?socket=/socket**

#### **-a nbd:///exportname?socket=/socket**

Add a disk located on Network Block Device (nbd) storage.

The *exportname* part of the URI specifies an NBD export name, but is usually left empty.

The optional *?socket* parameter can be used to specify a Unix domain socket that we talk to the NBD server



over. Note that you cannot mix server name (ie. TCP/IP) and socket path.

The equivalent API command would be (no export name):

```
><fs> add "" protocol:nbd server:[tcp:example.com|unix:/socket]
```

#### **-a rbd:///pool/disk**

#### **-a rbd://example.com[:port]/pool/disk**

Add a disk image located on a Ceph (RBD/librbd) storage volume.

Although libguestfs and Ceph supports multiple servers, only a single server can be specified when using this URI syntax.

The equivalent API command would be:

```
><fs> add pool/disk protocol:rbd server:tcp:example.com:port
```

#### **-a sheepdog://[example.com[:port]]/volume/image**

Add a disk image located on a Sheepdog volume.

The server name is optional. Although libguestfs and Sheepdog supports multiple servers, only at most one server can be specified when using this URI syntax.

The equivalent API command would be:

```
><fs> add volume protocol:sheepdog [server:tcp:example.com]
```

#### **-a ssh://[user@]example.com[:port]/disk.img**

Add a disk image located on a remote server, accessed using the Secure Shell (ssh) SFTP protocol. SFTP is supported out of the box by all major SSH servers.

The equivalent API command would be:

```
><fs> add /disk protocol:ssh server:tcp:example.com [username:user]
```

Note that the URIs follow the syntax of RFC 3986: in particular, there are restrictions on the allowed characters for the various components of the URI. Characters such as :, @, and / **must** be percent-encoded:

```
$ guestfish -a ssh://user:pass%40word@example.com/disk.img
```

In this case, the password is pass@word.

## **PROGRESS BARS**

Some (not all) long-running commands send progress notification messages as they are running. Guestfish turns these messages into progress bars.

When a command that supports progress bars takes longer than two seconds to run, and if progress bars are enabled, then you will see one appearing below the command:

```
><fs> copy-size /large-file /another-file 2048M
/ 10% [#####-----] 00:30
```

The spinner on the left hand side moves round once for every progress notification received from the backend. This is a (reasonably) golden assurance that the command is “doing something” even if the progress bar is not moving, because the command is able to send the progress notifications. When the bar reaches 100% and the command finishes, the spinner disappears.

Progress bars are enabled by default when guestfish is used interactively. You can enable them even for non-interactive modes using *--progress-bars*, and you can disable them completely using *--no-progress-bars*.

## **PROMPT**

You can change or add colours to the default prompt (><fs>) by setting the GUESTFISH\_PS1 environment variable. A second string (GUESTFISH\_OUTPUT) is printed after the command has been entered and before the output, allowing you to control the colour of the output. A third string (GUESTFISH\_INIT) is printed before the welcome message, allowing you to control the colour of that message. A fourth string (GUESTFISH\_RESTORE) is printed before guestfish exits.

A simple prompt can be set by setting GUESTFISH\_PS1 to an alternate string:

```
$ GUESTFISH_PS1='(type a command) '
$ export GUESTFISH_PS1
$ guestfish
[...]
(type a command)
```

You can also use special escape sequences, as described in the table below:

`\\` A literal backslash character.

`\[`

`\]` (These should only be used in GUESTFISH\_PS1.)

Place non-printing characters (eg. terminal control codes for colours) between `\[ . . \]`. What this does is to tell the **readline**(3) library that it should treat this subsequence as zero-width, so that command-line redisplay, editing etc works.

`\a` A bell character.

`\e` An ASCII ESC (escape) character.

`\n` A newline.

`\r` A carriage return.

`\NNN`

The ASCII character whose code is the octal value NNN.

`\xNN`

The ASCII character whose code is the hex value NN.

## EXAMPLES OF PROMPTS

Note that these examples require a terminal that supports ANSI escape codes.

- 

```
GUESTFISH_PS1='[\e[1;30m\]><fs>[\e[0;30m\] '
```

A bold black version of the ordinary prompt.

- 

```
GUESTFISH_PS1='[\e[1;32m\]><fs>[\e[0;31m\] '
GUESTFISH_OUTPUT='\e[0m'
GUESTFISH_RESTORE="$GUESTFISH_OUTPUT"
GUESTFISH_INIT='\e[1;34m'
```

Blue welcome text, green prompt, red commands, black command output.

## WINDOWS 8

Windows 8 “fast startup” can prevent guestfish from mounting NTFS partitions. See “WINDOWS HIBERNATION AND WINDOWS 8 FAST STARTUP” in **guestfs**(3).

## GUESTFISH COMMANDS

The commands in this section are guestfish convenience commands, in other words, they are not part of the **guestfs**(3) API.

### help

```
help
help cmd
help -l|--list
```

Without any parameter, this provides general help.

With a `cmd` parameter, this displays detailed help for that command.

With `-l` or `--list`, this list all commands.

### **exit**

### **quit**

This exits guestfish. You can also use `^D` key.

### **alloc**

### **allocate**

```
alloc filename size
```

This creates an empty (zeroed) file of the given size, and then adds so it can be further examined.

For more advanced image creation, see “disk-create”.

Size can be specified using standard suffixes, eg. 1M.

To create a sparse file, use “sparse” instead. To create a prepared disk image, see “PREPARED DISK IMAGES”.

### **copy-in**

```
copy-in local [local ...] /remotedir
```

`copy-in` copies local files or directories recursively into the disk image, placing them in the directory called `/remotedir` (which must exist). This guestfish meta-command turns into a sequence of “tar-in” and other commands as necessary.

Multiple local files and directories can be specified, but the last parameter must always be a remote directory. Wildcards cannot be used.

### **copy-out**

```
copy-out remote [remote ...] localdir
```

`copy-out` copies remote files or directories recursively out of the disk image, placing them on the host disk in a local directory called `localdir` (which must exist). This guestfish meta-command turns into a sequence of “download”, “tar-out” and other commands as necessary.

Multiple remote files and directories can be specified, but the last parameter must always be a local directory. To download to the current directory, use `.` as in:

```
copy-out /home .
```

Wildcards cannot be used in the ordinary command, but you can use them with the help of “glob” like this:

```
glob copy-out /home/* .
```

### **delete-event**

```
delete-event name
```

Delete the event handler which was previously registered as `name`. If multiple event handlers were registered with the same name, they are all deleted.

See also the guestfish commands `event` and `list-events`.

### **display**

```
display filename
```

Use `display` (a graphical display program) to display an image file. It downloads the file, and runs `display` on it.

To use an alternative program, set the `GUESTFISH_DISPLAY_IMAGE` environment variable. For example to use the GNOME display program:

```
export GUESTFISH_DISPLAY_IMAGE=eog
```

See also **display**(1).

**echo**

```
echo [params ...]
```

This echos the parameters to the terminal.

**edit****vi****emacs**

```
edit filename
```

This is used to edit a file. It downloads the file, edits it locally using your editor, then uploads the result.

The editor is \$EDITOR. However if you use the alternate commands `vi` or `emacs` you will get those corresponding editors.

**event**

```
event name eventset "shell script ..."
```

Register a shell script fragment which is executed when an event is raised. See “`guestfs_set_event_callback`” in **guestfs** (3) for a discussion of the event API in libguestfs.

The name parameter is a name that you give to this event handler. It can be any string (even the empty string) and is simply there so you can delete the handler using the `guestfish delete-event` command.

The eventset parameter is a comma-separated list of one or more events, for example `close` or `close,trace`. The special value `*` means all events.

The third and final parameter is the shell script fragment (or any external command) that is executed when any of the events in the eventset occurs. It is executed using `$SHELL -c`, or if `$SHELL` is not set then `/bin/sh -c`.

The shell script fragment receives callback parameters as arguments \$1, \$2 etc. The actual event that was called is available in the environment variable \$EVENT.

```
event "" close "echo closed"
event messages appliance,library,trace "echo $@"
event "" progress "echo progress: $3/$4"
event "" * "echo $EVENT $@"
```

See also the `guestfish` commands `delete-event` and `list-events`.

**glob**

```
glob command args...
```

Expand wildcards in any paths in the args list, and run command repeatedly on each matching path.

See “WILDCARDS AND GLOBBING”.

**hexedit**

```
hexedit <filename|device>
hexedit <filename|device> <max>
hexedit <filename|device> <start> <max>
```

Use hexedit (a hex editor) to edit all or part of a binary file or block device.

This command works by downloading potentially the whole file or device, editing it locally, then uploading it. If the file or device is large, you have to specify which part you wish to edit by using `max` and/or `start` `max` parameters. `start` and `max` are specified in bytes, with the usual modifiers allowed such as `1M` (1 megabyte).

For example to edit the first few sectors of a disk you might do:

```
hexedit /dev/sda 1M
```

which would allow you to edit anywhere within the first megabyte of the disk.

To edit the superblock of an ext2 filesystem on `/dev/sda1`, do:

```
hexedit /dev/sda1 0x400 0x400
```

(assuming the superblock is in the standard location).

This command requires the external **hexedit**(1) program. You can specify another program to use by setting the `HEXEDITOR` environment variable.

See also “hexdump”.

**lcd**

```
lcd directory
```

Change the local directory, ie. the current directory of guestfish itself.

Note that `!cd` won't do what you might expect.

**list-events**

```
list-events
```

List the event handlers registered using the guestfish event command.

**man****manual**

```
man
```

Opens the manual page for guestfish.

**more****less**

```
more filename
```

```
less filename
```

This is used to view a file.

The default viewer is `$PAGER`. However if you use the alternate command `less` you will get the `less` command specifically.

**reopen**

```
reopen
```

Close and reopen the `libguestfs` handle. It is not necessary to use this normally, because the handle is closed properly when guestfish exits. However this is occasionally useful for testing.

**setenv**

```
setenv VAR value
```

Set the environment variable `VAR` to the string `value`.

To print the value of an environment variable use a shell command such as:

```
!echo $VAR
```

**sparse**

```
sparse filename size
```

This creates an empty sparse file of the given size, and then adds so it can be further examined.

In all respects it works the same as the “`alloc`” command, except that the image file is allocated sparsely, which means that disk blocks are not assigned to the file until they are needed. Sparse disk files only use space when written to, but they are slower and there is a danger you could run out of real disk space during a write operation.

For more advanced image creation, see “`disk-create`”.

Size can be specified using standard suffixes, eg. `1M`.

See also the guestfish “`scratch`” command.

**supported**

supported

This command returns a list of the optional groups known to the daemon, and indicates which ones are supported by this build of the libguestfs appliance.

See also “AVAILABILITY” in **guestfs** (3).

**time**

time command args...

Run the command as usual, but print the elapsed time afterwards. This can be useful for benchmarking operations.

**unsetenv**

unsetenv VAR

Remove VAR from the environment.

**COMMANDS****acl-delete-def-file**

acl-delete-def-file dir

This function deletes the default POSIX Access Control List (ACL) attached to directory *dir*.

This command depends on the feature *acl*. See also “feature-available”.

**acl-get-file**

acl-get-file path acltype

This function returns the POSIX Access Control List (ACL) attached to *path*. The ACL is returned in “long text form” (see **acl** (5)).

The *acltype* parameter may be:

*access*

Return the ordinary (access) ACL for any file, directory or other filesystem object.

*default*

Return the default ACL. Normally this only makes sense if *path* is a directory.

This command depends on the feature *acl*. See also “feature-available”.

**acl-set-file**

acl-set-file path acltype acl

This function sets the POSIX Access Control List (ACL) attached to *path*.

The *acltype* parameter may be:

*access*

Set the ordinary (access) ACL for any file, directory or other filesystem object.

*default*

Set the default ACL. Normally this only makes sense if *path* is a directory.

The *acl* parameter is the new ACL in either “long text form” or “short text form” (see **acl** (5)). The new ACL completely replaces any previous ACL on the file. The ACL must contain the full Unix permissions (eg. *u::rwx,g::rwx,o::rwx*).

If you are specifying individual users or groups, then the mask field is also required (eg. *m::rwx*), followed by the *u:ID:...* and/or *g:ID:...* field(s). A full ACL string might therefore look like this:

```
u::rwx,g::rwx,o::rwx,m::rwx,u:500:rwx,g:500:rwx
\ Unix permissions / \mask/ \      ACL      /
```

You should use numeric UIDs and GIDs. To map usernames and groupnames to the correct numeric ID in the context of the guest, use the Augeas functions (see “aug-init”).

This command depends on the feature `acl`. See also “feature-available”.

### **add-cdrom**

```
add-cdrom filename
```

This function adds a virtual CD-ROM disk image to the guest.

The image is added as read-only drive, so this function is equivalent of “add-drive-ro”.

*This function is deprecated.* In new code, use the “add-drive-ro” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### **add-domain**

#### **domain**

```
add-domain dom [libvirturi:..] [readonly:true|false] [iface:..] [live:true|false]
```

This function adds the disk(s) attached to the named libvirt domain `dom`. It works by connecting to libvirt, requesting the domain and domain XML from libvirt, parsing it for disks, and calling “add-drive-opts” on each one.

The number of disks added is returned. This operation is atomic: if an error is returned, then no disks are added.

This function does some minimal checks to make sure the libvirt domain is not running (unless `readonly` is true). In a future version we will try to acquire the libvirt lock on each disk.

Disks must be accessible locally. This often means that adding disks from a remote libvirt connection (see <https://libvirt.org/remote.html>) will fail unless those disks are accessible via the same device path locally too.

The optional `libvirturi` parameter sets the libvirt URI (see <https://libvirt.org/uri.html>). If this is not set then we connect to the default libvirt URI (or one set through an environment variable, see the libvirt documentation for full details).

The optional `live` flag controls whether this call will try to connect to a running virtual machine `guestfsd` process if it sees a suitable `<channel>` element in the libvirt XML definition. The default (if the flag is omitted) is never to try. See “ATTACHING TO RUNNING DAEMONS” in **guestfs(3)** for more information.

If the `allowuuid` flag is true (default is false) then a UUID *may* be passed instead of the domain name. The `dom` string is treated as a UUID first and looked up, and if that lookup fails then we treat `dom` as a name as usual.

The optional `readonlydisk` parameter controls what we do for disks which are marked `<readonly/>` in the libvirt XML. Possible values are:

`readonlydisk = “error”`

If `readonly` is false:

The whole call is aborted with an error if any disk with the `<readonly/>` flag is found.

If `readonly` is true:

Disks with the `<readonly/>` flag are added read-only.

`readonlydisk = “read”`

If `readonly` is false:

Disks with the `<readonly/>` flag are added read-only. Other disks are added read/write.

If `readonly` is true:

Disks with the `<readonly/>` flag are added read-only.

`readonlydisk = "write"` (default)

If `readonly` is false:

Disks with the `<readonly/>` flag are added read/write.

If `readonly` is true:

Disks with the `<readonly/>` flag are added read-only.

`readonlydisk = "ignore"`

If `readonly` is true or false:

Disks with the `<readonly/>` flag are skipped.

If present, the value of `logical_block_size` attribute of `<blockio/>` tag in libvirt XML will be passed as `blocksize` parameter to `"add-drive-opts"`.

The other optional parameters are passed directly through to `"add-drive-opts"`.

This command has one or more optional arguments. See `"OPTIONAL ARGUMENTS"`.

### **add-drive**

#### **add**

#### **add-drive-opts**

```
add-drive filename [readonly:true|false] [format:...] [iface:...] [name:...] [label...]
```

This function adds a disk image called *filename* to the handle. *filename* may be a regular host file or a host device.

When this function is called before `"launch"` (the usual case) then the first time you call this function, the disk appears in the API as `/dev/sda`, the second time as `/dev/sdb`, and so on.

In `libguestfs`  $\geq 1.20$  you can also call this function after launch (with some restrictions). This is called `"hotplugging"`. When hotplugging, you must specify `label` so that the new disk gets a predictable name. For more information see `"HOTPLUGGING"` in `guestfs(3)`.

You don't necessarily need to be root when using `libguestfs`. However you obviously do need sufficient permissions to access the filename for whatever operations you want to perform (ie. read access if you just want to read the image or write access if you want to modify the image).

This call checks that *filename* exists.

*filename* may be the special string `"/dev/null"`. See `"NULL DISKS"` in `guestfs(3)`.

The optional arguments are:

#### `readonly`

If true then the image is treated as read-only. Writes are still allowed, but they are stored in a temporary snapshot overlay which is discarded at the end. The disk that you add is not modified.

#### `format`

This forces the image format. If you omit this (or use `"add-drive"` or `"add-drive-ro"`) then the format is automatically detected. Possible formats include `raw` and `qcow2`.

Automatic detection of the format opens you up to a potential security hole when dealing with untrusted raw-format images. See CVE-2010-3851 and RHBZ#642934. Specifying the format closes this security hole.

#### `iface`

This rarely-used option lets you emulate the behaviour of the deprecated `"add-drive-with-if"` call (q.v.)

#### `name`

The name the drive had in the original guest, e.g. `/dev/sdb`. This is used as a hint to the guest inspection process if it is available.



**label**

Give the disk a label. The label should be a unique, short string using *only* ASCII characters [a-zA-Z]. As well as its usual name in the API (such as `/dev/sda`), the drive will also be named `/dev/disk/guestfs/label`.

See “DISK LABELS” in **guestfs** (3).

**protocol**

The optional protocol argument can be used to select an alternate source protocol.

See also: “REMOTE STORAGE” in **guestfs** (3).

`protocol = ``file```

`filename` is interpreted as a local file or device. This is the default if the optional protocol parameter is omitted.

`protocol = ``ftp``|``ftps``|``http``|``https``|``tftp```

Connect to a remote FTP, HTTP or TFTP server. `Theserver` parameter must also be supplied – see below.

See also: “FTP, HTTP AND TFTP” in **guestfs** (3)

`protocol = ``gluster```

Connect to the GlusterFS server. `Theserver` parameter must also be supplied – see below.

See also: “GLUSTER” in **guestfs** (3)

`protocol = ``iscsi```

Connect to the iSCSI server. `Theserver` parameter must also be supplied – see below. The `username` parameter may be supplied. See below. `Thesecond` parameter may be supplied. See below.

See also: “ISCSI” in **guestfs** (3).

`protocol = ``nbd```

Connect to the Network Block Device server. `Theserver` parameter must also be supplied – see below.

See also: “NETWORK BLOCK DEVICE” in **guestfs** (3).

`protocol = ``rbd```

Connect to the Ceph (librbd/RBD) server. `Theserver` parameter must also be supplied – see below. The `username` parameter may be supplied. See below. `Thesecond` parameter may be supplied. See below.

See also: “CEPH” in **guestfs** (3).

`protocol = ``sheepdog```

Connect to the Sheepdog server. `Theserver` parameter may also be supplied – see below.

See also: “SHEEPDOG” in **guestfs** (3).

`protocol = ``ssh```

Connect to the Secure Shell (ssh) server.

The `server` parameter must be supplied. The `username` parameter may be supplied. See below.

See also: “SSH” in **guestfs** (3).

**server**

For protocols which require access to a remote server, this is a list of server(s).

Protocol	Number of servers required
-----	-----
file	List must be empty or param not used at all
ftp ftps http https tftp	Exactly one
gluster	Exactly one
iscsi	Exactly one
nbd	Exactly one
rbd	Zero or more
sheepdog	Zero or more
ssh	Exactly one

Each list element is a string specifying a server. The string must be in one of the following formats:

```
hostname
hostname:port
tcp:hostname
tcp:hostname:port
unix:/path/to/socket
```

If the port number is omitted, then the standard port number for the protocol is used (see */etc/services*).

#### username

For the ftp, ftps, http, https, iscsi, rbd, ssh and tftp protocols, this specifies the remote username.

If not given, then the local username is used for ssh, and no authentication is attempted for ceph. But note this sometimes may give unexpected results, for example if using the libvirt backend and if the libvirt backend is configured to start the qemu appliance as a special user such as `qemu.qemu`. If in doubt, specify the remote username you want.

#### secret

For the rbd protocol only, this specifies the ‘secret’ to use when connecting to the remote device. It must be base64 encoded.

If not given, then a secret matching the given username will be looked up in the default keychain locations, or if no username is given, then no authentication will be used.

#### cachemode

Choose whether or not libguestfs will obey sync operations (safe but slow) or not (unsafe but fast). The possible values for this string are:

```
cachemode = ``writeback``
```

This is the default.

Write operations in the API do not return until a **write**(2) call has completed in the host [but note this does not imply that anything gets written to disk].

Sync operations in the API, including implicit syncs caused by filesystem journalling, will not return until an **fdatasync**(2) call has completed in the host, indicating that data has been committed to disk.

```
cachemode = ``unsafe``
```

In this mode, there are no guarantees. Libguestfs may cache anything and ignore sync requests. This is suitable only for scratch or temporary disks.

#### discard

Enable or disable discard (a.k.a. trim or unmap) support on this drive. If enabled, operations such as “fstrim” will be able to discard / make thin / punch holes in the underlying host file or device.

Possible discard settings are:

```
discard = ``disable``
```

Disable discard support. This is the default.

```
discard = ``enable``
```

Enable discard support. Fail if discard is not possible.

```
discard = ``besteffort``
```

Enable discard support if possible, but don't fail if it is not supported.

Since not all backends and not all underlying systems support discard, this is a good choice if you want to use discard if possible, but don't mind if it doesn't work.

**copyonread**

The boolean parameter `copyonread` enables copy-on-read support. This only affects disk formats which have backing files, and causes reads to be stored in the overlay layer, speeding up multiple reads of the same area of disk.

The default is false.

**blocksize**

This parameter sets the sector size of the disk. Possible values are 512 (the default if the parameter is omitted) or 4096. Use 4096 when handling an 'Advanced Format' disk that uses 4K sector size ([https://en.wikipedia.org/wiki/Advanced\\_Format](https://en.wikipedia.org/wiki/Advanced_Format)).

Only a subset of the backends support this parameter (currently only the libvirt and direct backends do).

This command has one or more optional arguments. See "OPTIONAL ARGUMENTS".

## **add-drive-ro**

### **add-ro**

```
add-drive-ro filename
```

This function is the equivalent of calling "add-drive-opts" with the optional parameter `GUESTFS_ADD_DRIVE_OPTS_READONLY` set to 1, so the disk is added read-only, with the format being detected automatically.

## **add-drive-ro-with-if**

```
add-drive-ro-with-if filename iface
```

This is the same as "add-drive-ro" but it allows you to specify the QEMU interface emulation to use at run time.

*This function is deprecated.* In new code, use the "add-drive" call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

## **add-drive-scratch**

### **scratch**

```
add-drive-scratch size [name:...] [label:...] [blocksize:N]
```

This command adds a temporary scratch drive to the handle. The `size` parameter is the virtual size (in bytes). The scratch drive is blank initially (all reads return zeroes until you start writing to it). The drive is deleted when the handle is closed.

The optional arguments `name`, `label` and `blocksize` are passed through to "add-drive-opts".

This command has one or more optional arguments. See "OPTIONAL ARGUMENTS".

## **add-drive-with-if**

```
add-drive-with-if filename iface
```

This is the same as "add-drive" but it allows you to specify the QEMU interface emulation to use at run time.

*This function is deprecated.* In new code, use the "add-drive" call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### **aug-clear**

```
aug-clear augpath
```

Set the value associated with `path` to `NULL`. This is the same as the **augtool**(1) `clear` command.

### **aug-close**

```
aug-close
```

Close the current Augeas handle and free up any resources used by it. After calling this, you have to call “`aug-init`” again before you can use any other Augeas functions.

### **aug-defnode**

```
aug-defnode name expr val
```

Defines a variable `name` whose value is the result of evaluating `expr`.

If `expr` evaluates to an empty nodeset, a node is created, equivalent to calling “`aug-set`” `expr`, `val`. `name` will be the nodeset containing that single node.

On success this returns a pair containing the number of nodes in the nodeset, and a boolean flag if a node was created.

### **aug-defvar**

```
aug-defvar name expr
```

Defines an Augeas variable `name` whose value is the result of evaluating `expr`. If `expr` is `NULL`, then `name` is undefined.

On success this returns the number of nodes in `expr`, or 0 if `expr` evaluates to something which is not a nodeset.

### **aug-get**

```
aug-get augpath
```

Look up the value associated with `path`. If `path` matches exactly one node, the value is returned.

### **aug-init**

```
aug-init root flags
```

Create a new Augeas handle for editing configuration files. If there was any previous Augeas handle associated with this guestfs session, then it is closed.

You must call this before using any other “`aug-*`” commands.

`root` is the filesystem root. `root` must not be `NULL`, use `/` instead.

The flags are the same as the flags defined in `<augeas.h>`, the logical *or* of the following integers:

```
AUG_SAVE_BACKUP = 1
```

Keep the original file with a `.augsave` extension.

```
AUG_SAVE_NEWFILE = 2
```

Save changes into a file with extension `.augnew`, and do not overwrite original. Overrides `AUG_SAVE_BACKUP`.

```
AUG_TYPE_CHECK = 4
```

Typecheck lenses.

This option is only useful when debugging Augeas lenses. Use of this option may require additional memory for the libguestfs appliance. You may need to set the `LIBGUESTFS_MEMSIZE` environment variable or call “`set-memsize`”.

```
AUG_NO_STDINC = 8
```

Do not use standard load path for modules.

`AUG_SAVE_NOOP = 16`

Make save a no-op, just record what would have been changed.

`AUG_NO_LOAD = 32`

Do not load the tree in “aug-init”.

To close the handle, you can call “aug-close”.

To find out more about Augeas, see <http://augeas.net/>.

#### **aug-insert**

`aug-insert augpath label true|false`

Create a new sibling `label` for `path`, inserting it into the tree before or after `path` (depending on the boolean flag before).

`path` must match exactly one existing node in the tree, and `label` must be a label, ie. not contain `/`, `*` or end with a bracketed index `[N]`.

#### **aug-label**

`aug-label augpath`

The label (name of the last element) of the Augeas path expression `augpath` is returned. `augpath` must match exactly one node, else this function returns an error.

#### **aug-load**

`aug-load`

Load files into the tree.

See `aug_load` in the Augeas documentation for the full gory details.

#### **aug-ls**

`aug-ls augpath`

This is just a shortcut for listing “aug-match” `path/*` and sorting the resulting nodes into alphabetical order.

#### **aug-match**

`aug-match augpath`

Returns a list of paths which match the path expression `path`. The returned paths are sufficiently qualified so that they match exactly one node in the current tree.

#### **aug-mv**

`aug-mv src dest`

Move the node `src` to `dest`. `src` must match exactly one node. `dest` is overwritten if it exists.

#### **aug-rm**

`aug-rm augpath`

Remove `path` and all of its children.

On success this returns the number of entries which were removed.

#### **aug-save**

`aug-save`

This writes all pending changes to disk.

The flags which were passed to “aug-init” affect exactly how files are saved.

#### **aug-set**

`aug-set augpath val`

Set the value associated with `augpath` to `val`.

In the Augeas API, it is possible to clear a node by setting the value to NULL. Due to an oversight in the libguestfs API you cannot do that with this call. Instead you must use the “aug-clear” call.

**aug-setm**

```
aug-setm base sub val
```

Change multiple Augeas nodes in a single operation. `base` is an expression matching multiple nodes. `sub` is a path expression relative to `base`. All nodes matching `base` are found, and then for each node, `sub` is changed to `val`. `sub` may also be `NULL` in which case the `base` nodes are modified.

This returns the number of nodes modified.

**aug-transform**

```
aug-transform lens file [remove:true|false]
```

Add an Augeas transformation for the specified lens so it can handle `file`.

If `remove` is `true` (`false` by default), then the transformation is removed.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

**available**

```
available 'groups ...'
```

This command is used to check the availability of some groups of functionality in the appliance, which not all builds of the libguestfs appliance will be able to provide.

The libguestfs groups, and the functions that those groups correspond to, are listed in “AVAILABILITY” in **guestfs**(3). You can also fetch this list at runtime by calling “available-all-groups”.

The argument `groups` is a list of group names, eg: [ "inotify", "augeas" ] would check for the availability of the Linux inotify functions and Augeas (configuration file editing) functions.

The command returns no error if *all* requested groups are available.

It fails with an error if one or more of the requested groups is unavailable in the appliance.

If an unknown group name is included in the list of groups then an error is always returned.

*Notes:*

- “feature-available” is the same as this call, but with a slightly simpler to use API: that call returns a boolean true/false instead of throwing an error.
- You must call “launch” before calling this function.  
  
The reason is because we don’t know what groups are supported by the appliance/daemon until it is running and can be queried.
- If a group of functions is available, this does not necessarily mean that they will work. You still have to check for errors when calling individual API functions even if they are available.
- It is usually the job of distro packagers to build complete functionality into the libguestfs appliance. Upstream libguestfs, if built from source with all requirements satisfied, will support everything.
- This call was added in version 1.0.80. In previous versions of libguestfs all you could do would be to speculatively execute a command to find out if the daemon implemented it. See also “version”.

See also “filesystem-available”.

**available-all-groups**

```
available-all-groups
```

This command returns a list of all optional groups that this daemon knows about. Note this returns both supported and unsupported groups. To find out which ones the daemon can actually support you have to call “available” / “feature-available” on each member of the returned list.

See also “available”, “feature-available” and “AVAILABILITY” in **guestfs**(3).

**base64-in**

```
base64-in (base64file|-) filename
```

This command uploads base64-encoded data from `base64file` to `filename`.

Use `-` instead of a filename to read/write from stdin/stdout.

### **base64-out**

```
base64-out filename (base64file|-)
```

This command downloads the contents of *filename*, writing it out to local file *base64file* encoded as base64.

Use `-` instead of a filename to read/write from stdin/stdout.

### **blkdiscard**

```
blkdiscard device
```

This discards all blocks on the block device *device*, giving the free space back to the host.

This operation requires support in libguestfs, the host filesystem, qemu and the host kernel. If this support isn't present it may give an error or even appear to run but do nothing. You must also set the *discard* attribute on the underlying drive (see “add-drive-opts”).

This command depends on the feature *blkdiscard*. See also “feature-available”.

### **blkdiscardzeroes**

```
blkdiscardzeroes device
```

This call returns true if blocks on *device* that have been discarded by a call to “*blkdiscard*” are returned as blocks of zero bytes when read the next time.

If it returns false, then it may be that discarded blocks are read as stale or random data.

This command depends on the feature *blkdiscardzeroes*. See also “feature-available”.

### **blkid**

```
blkid device
```

This command returns block device attributes for *device*. The following fields are usually present in the returned hash. Other fields may also be present.

UUID

The uuid of this device.

LABEL

The label of this device.

VERSION

The version of *blkid* command.

TYPE

The filesystem type or RAID of this device.

USAGE

The usage of this device, for example *filesystem* or *raid*.

### **blockdev-flushbufs**

```
blockdev-flushbufs device
```

This tells the kernel to flush internal buffers associated with *device*.

This uses the **blockdev** (8) command.

### **blockdev-getbsz**

```
blockdev-getbsz device
```

This returns the block size of a device.

Note: this is different from both *size in blocks* and *filesystem block size*. Also this setting is not really used by anything. You should probably not use it for anything. Filesystems have their own idea about what block size to choose.

This uses the **blockdev** (8) command.

**blockdev-getro**

```
blockdev-getro device
```

Returns a boolean indicating if the block device is read-only (true if read-only, false if not).

This uses the **blockdev** (8) command.

**blockdev-getsize64**

```
blockdev-getsize64 device
```

This returns the size of the device in bytes.

See also “blockdev-getsz”.

This uses the **blockdev** (8) command.

**blockdev-getss**

```
blockdev-getss device
```

This returns the size of sectors on a block device. Usually 512, but can be larger for modern devices.

(Note, this is not the size in sectors, use “blockdev-getsz” for that).

This uses the **blockdev** (8) command.

**blockdev-getsz**

```
blockdev-getsz device
```

This returns the size of the device in units of 512-byte sectors (even if the sectorsize isn’t 512 bytes ... weird).

See also “blockdev-getss” for the real sector size of the device, and “blockdev-getsize64” for the more useful *size in bytes*.

This uses the **blockdev** (8) command.

**blockdev-rereadpt**

```
blockdev-rereadpt device
```

Reread the partition table on device.

This uses the **blockdev** (8) command.

**blockdev-setbsz**

```
blockdev-setbsz device blocksize
```

This call does nothing and has never done anything because of a bug in blockdev. **Do not use it.**

If you need to set the filesystem block size, use the `blocksize` option of “mkfs”.

*This function is deprecated.* There is no replacement. Consult the API documentation in **guestfs** (3) for further information.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

**blockdev-setra**

```
blockdev-setra device sectors
```

Set readahead (in 512-byte sectors) for the device.

This uses the **blockdev** (8) command.

**blockdev-setro**

```
blockdev-setro device
```

Sets the block device named `device` to read-only.

This uses the **blockdev** (8) command.



**blockdev-setrw**

```
blockdev-setrw device
```

Sets the block device named `device` to read-write.

This uses the **blockdev** (8) command.

**btrfs-balance-cancel**

```
btrfs-balance-cancel path
```

Cancel a running balance on a btrfs filesystem.

This command depends on the feature `btrfs`. See also “feature-available”.

**btrfs-balance-pause**

```
btrfs-balance-pause path
```

Pause a running balance on a btrfs filesystem.

This command depends on the feature `btrfs`. See also “feature-available”.

**btrfs-balance-resume**

```
btrfs-balance-resume path
```

Resume a paused balance on a btrfs filesystem.

This command depends on the feature `btrfs`. See also “feature-available”.

**btrfs-balance-status**

```
btrfs-balance-status path
```

Show the status of a running or paused balance on a btrfs filesystem.

This command depends on the feature `btrfs`. See also “feature-available”.

**btrfs-device-add**

```
btrfs-device-add 'devices ...' fs
```

Add the list of device(s) in `devices` to the btrfs filesystem mounted at `fs`. If `devices` is an empty list, this does nothing.

This command depends on the feature `btrfs`. See also “feature-available”.

**btrfs-device-delete**

```
btrfs-device-delete 'devices ...' fs
```

Remove the devices from the btrfs filesystem mounted at `fs`. If `devices` is an empty list, this does nothing.

This command depends on the feature `btrfs`. See also “feature-available”.

**btrfs-filesystem-balance****btrfs-balance**

```
btrfs-filesystem-balance fs
```

Balance the chunks in the btrfs filesystem mounted at `fs` across the underlying devices.

This command depends on the feature `btrfs`. See also “feature-available”.

**btrfs-filesystem-defragment**

```
btrfs-filesystem-defragment path [flush:true|false] [compress:...]
```

Defragment a file or directory on a btrfs filesystem. `compress` is one of `zlib` or `lzo`.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

This command depends on the feature `btrfs`. See also “feature-available”.

**btrfs-filesystem-resize**

```
btrfs-filesystem-resize mountpoint [size:N]
```

This command resizes a btrfs filesystem.

Note that unlike other resize calls, the filesystem has to be mounted and the parameter is the mountpoint not the device (this is a requirement of btrfs itself).

The optional parameters are:

`size`

The new size (in bytes) of the filesystem. If omitted, the filesystem is resized to the maximum size.

See also **btrfs** (8).

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

This command depends on the feature `btrfs`. See also “feature-available”.

#### **btrfs-filesystem-show**

`btrfs-filesystem-show device`

Show all the devices where the filesystems in `device` is spanned over.

If not all the devices for the filesystems are present, then this function fails and the `errno` is set to `ENODEV`.

This command depends on the feature `btrfs`. See also “feature-available”.

#### **btrfs-filesystem-sync**

`btrfs-filesystem-sync fs`

Force sync on the btrfs filesystem mounted at `fs`.

This command depends on the feature `btrfs`. See also “feature-available”.

#### **btrfs-fsck**

`btrfs-fsck device [superblock:N] [repair:true|false]`

Used to check a btrfs filesystem, `device` is the device file where the filesystem is stored.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

This command depends on the feature `btrfs`. See also “feature-available”.

#### **btrfs-image**

`btrfs-image 'source ...' image [compresslevel:N]`

This is used to create an image of a btrfs filesystem. All data will be zeroed, but metadata and the like is preserved.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

This command depends on the feature `btrfs`. See also “feature-available”.

#### **btrfs-qgroup-assign**

`btrfs-qgroup-assign src dst path`

Add qgroup `src` to parent qgroup `dst`. This command can group several qgroups into a parent qgroup to share common limit.

This command depends on the feature `btrfs`. See also “feature-available”.

#### **btrfs-qgroup-create**

`btrfs-qgroup-create qgroupid subvolume`

Create a quota group (qgroup) for subvolume at `subvolume`.

This command depends on the feature `btrfs`. See also “feature-available”.

#### **btrfs-qgroup-destroy**

`btrfs-qgroup-destroy qgroupid subvolume`

Destroy a quota group.

This command depends on the feature `btrfs`. See also “feature-available”.

**btrfs-qgroup-limit**

```
btrfs-qgroup-limit subvolume size
```

Limit the size of the subvolume with path subvolume.

This command depends on the feature `btrfs`. See also “feature-available”.

**btrfs-qgroup-remove**

```
btrfs-qgroup-remove src dst path
```

Remove qgroup `src` from the parent qgroup `dst`.

This command depends on the feature `btrfs`. See also “feature-available”.

**btrfs-qgroup-show**

```
btrfs-qgroup-show path
```

Show all subvolume quota groups in a btrfs filesystem, including their usages.

This command depends on the feature `btrfs`. See also “feature-available”.

**btrfs-quota-enable**

```
btrfs-quota-enable fs true|false
```

Enable or disable subvolume quota support for filesystem which contains path.

This command depends on the feature `btrfs`. See also “feature-available”.

**btrfs-quota-rescan**

```
btrfs-quota-rescan fs
```

Trash all qgroup numbers and scan the metadata again with the current config.

This command depends on the feature `btrfs`. See also “feature-available”.

**btrfs-replace**

```
btrfs-replace srcdev targetdev mntpoint
```

Replace device of a btrfs filesystem. On a live filesystem, duplicate the data to the target device which is currently stored on the source device. After completion of the operation, the source device is wiped out and removed from the filesystem.

The `targetdev` needs to be same size or larger than the `srcdev`. Devices which are currently mounted are never allowed to be used as the `targetdev`.

This command depends on the feature `btrfs`. See also “feature-available”.

**btrfs-rescue-chunk-recover**

```
btrfs-rescue-chunk-recover device
```

Recover the chunk tree of btrfs filesystem by scanning the devices one by one.

This command depends on the feature `btrfs`. See also “feature-available”.

**btrfs-rescue-super-recover**

```
btrfs-rescue-super-recover device
```

Recover bad superblocks from good copies.

This command depends on the feature `btrfs`. See also “feature-available”.

**btrfs-scrub-cancel**

```
btrfs-scrub-cancel path
```

Cancel a running scrub on a btrfs filesystem.

This command depends on the feature `btrfs`. See also “feature-available”.

**btrfs-scrub-resume**

```
btrfs-scrub-resume path
```

Resume a previously canceled or interrupted scrub on a btrfs filesystem.

This command depends on the feature `btrfs`. See also “feature-available”.

**btrfs-scrub-start**

```
btrfs-scrub-start path
```

Reads all the data and metadata on the filesystem, and uses checksums and the duplicate copies from RAID storage to identify and repair any corrupt data.

This command depends on the feature `btrfs`. See also “feature-available”.

**btrfs-scrub-status**

```
btrfs-scrub-status path
```

Show status of running or finished scrub on a `btrfs` filesystem.

This command depends on the feature `btrfs`. See also “feature-available”.

**btrfs-set-seeding**

```
btrfs-set-seeding device true|false
```

Enable or disable the seeding feature of a device that contains a `btrfs` filesystem.

This command depends on the feature `btrfs`. See also “feature-available”.

**btrfs-subvolume-create****btrfs-subvolume-create-opts**

```
btrfs-subvolume-create dest [qgroupid:..]
```

Create a `btrfs` subvolume. The `dest` argument is the destination directory and the name of the subvolume, in the form `/path/to/dest/name`. The optional parameter `qgroupid` represents the `qgroup` which the newly created subvolume will be added to.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

This command depends on the feature `btrfs`. See also “feature-available”.

**btrfs-subvolume-delete**

```
btrfs-subvolume-delete subvolume
```

Delete the named `btrfs` subvolume or snapshot.

This command depends on the feature `btrfs`. See also “feature-available”.

**btrfs-subvolume-get-default**

```
btrfs-subvolume-get-default fs
```

Get the default subvolume or snapshot of a filesystem mounted at `mountpoint`.

This command depends on the feature `btrfs`. See also “feature-available”.

**btrfs-subvolume-list**

```
btrfs-subvolume-list fs
```

List the `btrfs` snapshots and subvolumes of the `btrfs` filesystem which is mounted at `fs`.

This command depends on the feature `btrfs`. See also “feature-available”.

**btrfs-subvolume-set-default**

```
btrfs-subvolume-set-default id fs
```

Set the subvolume of the `btrfs` filesystem `fs` which will be mounted by default. See “`btrfs-subvolume-list`” to get a list of subvolumes.

This command depends on the feature `btrfs`. See also “feature-available”.

**btrfs-subvolume-show**

```
btrfs-subvolume-show subvolume
```

Return detailed information of the subvolume.

This command depends on the feature `btrfs`. See also “feature-available”.

**btrfs-subvolume-snapshot****btrfs-subvolume-snapshot-opts**

```
btrfs-subvolume-snapshot source dest [ro:true|false] [qgroupid:..]
```

Create a snapshot of the btrfs subvolume *source*. The *dest* argument is the destination directory and the name of the snapshot, in the form */path/to/dest/name*. By default the newly created snapshot is writable, if the value of optional parameter *ro* is true, then a readonly snapshot is created. The optional parameter *qgroupid* represents the qgroup which the newly created snapshot will be added to.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

This command depends on the feature *btrfs*. See also “feature-available”.

**btrfstune-enable-extended-inode-refs**

```
btrfstune-enable-extended-inode-refs device
```

This will Enable extended inode refs.

This command depends on the feature *btrfs*. See also “feature-available”.

**btrfstune-enable-skinny-metadata-extent-refs**

```
btrfstune-enable-skinny-metadata-extent-refs device
```

This enable skinny metadata extent refs.

This command depends on the feature *btrfs*. See also “feature-available”.

**btrfstune-seeding**

```
btrfstune-seeding device true|false
```

Enable seeding of a btrfs device, this will force a fs readonly so that you can use it to build other filesystems.

This command depends on the feature *btrfs*. See also “feature-available”.

**c-pointer**

```
c-pointer
```

In non-C language bindings, this allows you to retrieve the underlying C pointer to the handle (ie. “h \*”). The purpose of this is to allow other libraries to interwork with libguestfs.

**canonical-device-name**

```
canonical-device-name device
```

This utility function is useful when displaying device names to the user. It takes a number of irregular device names and returns them in a consistent format:

```
/dev/hdX
```

```
/dev/vdX
```

These are returned as */dev/sdX*. Note this works for device names and partition names. This is approximately the reverse of the algorithm described in “BLOCK DEVICE NAMING” in **guestfs** (3).

```
/dev/mapper/VG-LV
```

```
/dev/dm-N
```

Converted to */dev/VG/LV* form using “lvm-canonical-lv-name”.

Other strings are returned unmodified.

**cap-get-file**

```
cap-get-file path
```

This function returns the Linux capabilities attached to *path*. The capabilities set is returned in text form (see **cap\_to\_text** (3)).

If no capabilities are attached to a file, an empty string is returned.

This command depends on the feature *linuxcaps*. See also “feature-available”.

**cap-set-file**

```
cap-set-file path cap
```

This function sets the Linux capabilities attached to `path`. The capabilities `setcap` should be passed in text form (see `cap_from_text(3)`).

This command depends on the feature `linuxcaps`. See also “feature-available”.

**case-sensitive-path**

```
case-sensitive-path path
```

This can be used to resolve case insensitive paths on a filesystem which is case sensitive. The use case is to resolve paths which you have read from Windows configuration files or the Windows Registry, to the true path.

The command handles a peculiarity of the Linux `ntfs-3g` filesystem driver (and probably others), which is that although the underlying filesystem is case-insensitive, the driver exports the filesystem to Linux as case-sensitive.

One consequence of this is that special directories such as `C:\windows` may appear as `/WINDOWS` or `/windows` (or other things) depending on the precise details of how they were created. In Windows itself this would not be a problem.

Bug or feature? You decide: <https://www.tuxera.com/community/ntfs-3g-faq/#posixfilenames1>

“case-sensitive-path” attempts to resolve the true case of each element in the path. It will return a resolved path if either the full path or its parent directory exists. If the parent directory exists but the full path does not, the case of the parent directory will be correctly resolved, and the remainder appended unmodified. For example, if the file `" /Windows/System32/netkvm.sys "` exists:

```
“case-sensitive-path” (“/windows/system32/netkvm.sys”)
```

```
“Windows/System32/netkvm.sys”
```

```
“case-sensitive-path” (“/windows/system32/NoSuchFile”)
```

```
“Windows/System32/NoSuchFile”
```

```
“case-sensitive-path” (“/windows/system33/netkvm.sys”)
```

```
ERROR
```

*Note:* Because of the above behaviour, “case-sensitive-path” cannot be used to check for the existence of a file.

*Note:* This function does not handle drive names, backslashes etc.

See also “realpath”.

**cat**

```
cat path
```

Return the contents of the file named `path`.

Because, in C, this function returns a `char *`, there is no way to differentiate between a `\0` character in a file and end of string. To handle binary files, use the “read-file” or “download” functions.

**checksum**

```
checksum csumtype path
```

This call computes the MD5, SHAx or CRC checksum of the file named `path`.

The type of checksum to compute is given by the `csumtype` parameter which must have one of the following values:

```
crc
```

Compute the cyclic redundancy check (CRC) specified by POSIX for the `cksum` command.

```
md5
```

Compute the MD5 hash (using the `md5sum(1)` program).

**sha1**

Compute the SHA1 hash (using the **sha1sum** (1) program).

**sha224**

Compute the SHA224 hash (using the **sha224sum** (1) program).

**sha256**

Compute the SHA256 hash (using the **sha256sum** (1) program).

**sha384**

Compute the SHA384 hash (using the **sha384sum** (1) program).

**sha512**

Compute the SHA512 hash (using the **sha512sum** (1) program).

The checksum is returned as a printable string.

To get the checksum for a device, use “checksum-device”.

To get the checksums for many files, use “checksums-out”.

**checksum-device**

```
checksum-device csumtype device
```

This call computes the MD5, SHAx or CRC checksum of the contents of the device named *device*. For the types of checksums supported see the “checksum” command.

**checksums-out**

```
checksums-out csumtype directory (sumsfile|-)
```

This command computes the checksums of all regular files in *directory* and then emits a list of those checksums to the local output file *sumsfile*.

This can be used for verifying the integrity of a virtual machine. However to be properly secure you should pay attention to the output of the checksum command (it uses the ones from GNU coreutils). In particular when the filename is not printable, coreutils uses a special backslash syntax. For more information, see the GNU coreutils info file.

Use *-* instead of a filename to read/write from stdin/stdout.

**chmod**

```
chmod mode path
```

Change the mode (permissions) of *path* to *mode*. Only numeric modes are supported.

*Note:* When using this command from guestfish, *mode* by default would be decimal, unless you prefix it with 0 to get octal, ie. use 0700 not 700.

The mode actually set is affected by the umask.

**chown**

```
chown owner group path
```

Change the file owner to *owner* and group to *group*.

Only numeric uid and gid are supported. If you want to use names, you will need to locate and parse the password file yourself (Augeas support makes this relatively easy).

**clear-backend-setting**

```
clear-backend-setting name
```

If there is a backend setting string matching “*name*” or beginning with “*name*=”, then that string is removed from the backend settings.

This call returns the number of strings which were removed (which may be 0, 1 or greater than 1).

See “BACKEND” in **guestfs** (3), “BACKEND SETTINGS” in **guestfs** (3).

**command**

```
command 'arguments ...'
```

This call runs a command from the guest filesystem. The filesystem must be mounted, and must contain a compatible operating system (ie. something Linux, with the same or compatible processor architecture).

The single parameter is an argv-style list of arguments. The first element is the name of the program to run. Subsequent elements are parameters. The list must be non-empty (ie. must contain a program name). Note that the command runs directly, and is *not* invoked via the shell (see “sh”).

The return value is anything printed to *stdout* by the command.

If the command returns a non-zero exit status, then this function returns an error message. The error message string is the content of *stderr* from the command.

The `$PATH` environment variable will contain at least `/usr/bin` and `/bin`. If you require a program from another location, you should provide the full path in the first parameter.

Shared libraries and data files required by the program must be available on filesystems which are mounted in the correct places. It is the caller’s responsibility to ensure all filesystems that are needed are mounted at the right locations.

Because of the message protocol, there is a transfer limit of somewhere between 2MB and 4MB. See “PROTOCOL LIMITS” in **guestfs** (3).

**command-lines**

```
command-lines 'arguments ...'
```

This is the same as “command”, but splits the result into a list of lines.

See also: “sh-lines”

Because of the message protocol, there is a transfer limit of somewhere between 2MB and 4MB. See “PROTOCOL LIMITS” in **guestfs** (3).

**compress-device-out**

```
compress-device-out ctype device (zdevice|-) [level:N]
```

This command compresses *device* and writes it out to the local file *zdevice*.

The *ctype* and optional *level* parameters have the same meaning as in “compress-out”.

Use `-` instead of a filename to read/write from *stdin*/*stdout*.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

**compress-out**

```
compress-out ctype file (zfile|-) [level:N]
```

This command compresses *file* and writes it out to the local file *zfile*.

The compression program used is controlled by the *ctype* parameter. Currently this includes: `compress`, `gzip`, `bzip2`, `xz` or `lzop`. Some compression types may not be supported by particular builds of `libguestfs`, in which case you will get an error containing the substring “not supported”.

The optional *level* parameter controls compression level. The meaning and default for this parameter depends on the compression program being used.

Use `-` instead of a filename to read/write from *stdin*/*stdout*.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

**config**

```
config hvparam hvvalue
```

This can be used to add arbitrary hypervisor parameters of the form `-param value`. Actually it’s not quite arbitrary – we prevent you from setting some parameters which would interfere with parameters that we use.



The first character of `hvparam` string must be a `-` (dash).

`hvvalue` can be `NULL`.

### copy-attributes

```
copy-attributes src dest [all:true|false] [mode:true|false] [xattributes:true|fa
```

Copy the attributes of a path (which can be a file or a directory) to another path.

By default **no** attribute is copied, so make sure to specify any (or `all` to copy everything).

The optional arguments specify which attributes can be copied:

`mode`

Copy part of the file mode from source to destination. Only the UNIX permissions and the sticky/setuid/setgid bits can be copied.

`xattributes`

Copy the Linux extended attributes (`xattrs`) from source to destination. This flag does nothing if the `linuxxattrs` feature is not available (see “feature-available”).

`ownership`

Copy the owner uid and the group gid of source to destination.

`all`

Copy **all** the attributes from source to destination. Enabling it enables all the other flags, if they are not specified already.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

### copy-device-to-device

```
copy-device-to-device src dest [srcoffset:N] [destoffset:N] [size:N] [sparse:true
```

The four calls “copy-device-to-device”, “copy-device-to-file”, “copy-file-to-device”, and “copy-file-to-file” let you copy from a source (device|file) to a destination (device|file).

Partial copies can be made since you can specify optionally the source offset, destination offset and size to copy. These values are all specified in bytes. If not given, the offsets both default to zero, and the size defaults to copying as much as possible until we hit the end of the source.

The source and destination may be the same object. However overlapping regions may not be copied correctly.

If the destination is a file, it is created if required. If the destination file is not large enough, it is extended.

If the destination is a file and the `append` flag is not set, then the destination file is truncated. If the `append` flag is set, then the copy appends to the destination file. The `append` flag currently cannot be set for devices.

If the `sparse` flag is true then the call avoids writing blocks that contain only zeroes, which can help in some situations where the backing disk is thin-provisioned. Note that unless the target is already zeroed, using this option will result in incorrect copying.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

### copy-device-to-file

```
copy-device-to-file src dest [srcoffset:N] [destoffset:N] [size:N] [sparse:true|
```

See “copy-device-to-device” for a general overview of this call.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

### copy-file-to-device

```
copy-file-to-device src dest [srcoffset:N] [destoffset:N] [size:N] [sparse:true|
```

See “copy-device-to-device” for a general overview of this call.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

**copy-file-to-file**

```
copy-file-to-file src dest [srcoffset:N] [destoffset:N] [size:N] [sparse:true|false]
```

See “copy-device-to-device” for a general overview of this call.

This is **not** the function you want for copying files. This is for copying blocks within existing files. See “cp”, “cp-a” and “mv” for general file copying and moving functions.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

**copy-size**

```
copy-size src dest size
```

This command copies exactly `size` bytes from one source device or file `src` to another destination device or file `dest`.

Note this will fail if the source is too short or if the destination is not large enough.

*This function is deprecated.* In new code, use the “copy-device-to-device” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

**cp**

```
cp src dest
```

This copies a file from `src` to `dest` where `dest` is either a destination filename or destination directory.

**cp-a**

```
cp-a src dest
```

This copies a file or directory from `src` to `dest` recursively using the `cp -a` command.

**cp-r**

```
cp-r src dest
```

This copies a file or directory from `src` to `dest` recursively using the `cp -rP` command.

Most users should use “cp-a” instead. This command is useful when you don’t want to preserve permissions, because the target filesystem does not support it (primarily when writing to DOS FAT filesystems).

**cpio-out**

```
cpio-out directory (cpiofile|-) [format:..]
```

This command packs the contents of *directory* and downloads it to local file *cpiofile*.

The optional `format` parameter can be used to select the format. Only the following formats are currently permitted:

**newc**

New (SVR4) portable format. This format happens to be compatible with the cpio-like format used by the Linux kernel for `initramfs`.

This is the default format.

**crc**

New (SVR4) portable format with a checksum.

Use `-` instead of a filename to read/write from `stdin/stdout`.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

**cryptsetup-close**

```
cryptsetup-close device
```

This closes an encrypted device that was created earlier by “cryptsetup-open”. The `device` parameter must be the name of the mapping device (ie. `/dev/mapper/mapname`) and *not* the name of the underlying block device.

This command depends on the feature `luks`. See also “feature-available”.

### **cryptsetup-open**

```
cryptsetup-open device mapname [readonly:true|false] [crypttype:..]
```

This command opens a block device which has been encrypted according to the Linux Unified Key Setup (LUKS) standard, Windows BitLocker, or some other types.

`device` is the encrypted block device or partition.

The caller must supply one of the keys associated with the encrypted block device, in the `key` parameter.

This creates a new block device called `/dev/mapper/mapname`. Reads and writes to this block device are decrypted from and encrypted to the underlying device respectively.

`mapname` cannot be `"control"` because that name is reserved by device-mapper.

If the optional `crypttype` parameter is not present then `libguestfs` tries to guess the correct type (for example LUKS or BitLocker). However you can override this by specifying one of the following types:

`luks`

A Linux LUKS device.

`bitlk`

A Windows BitLocker device.

The optional `readonly` flag, if set to `true`, creates a read-only mapping.

If this block device contains LVM volume groups, then calling “`lvm-scan`” with the `activate` parameter `true` will make them visible.

Use “`list-dm-devices`” to list all device mapper devices.

This command has one or more key or passphrase parameters. `Guestfish` will prompt for these separately.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

This command depends on the feature `luks`. See also “feature-available”.

### **dd**

```
dd src dest
```

This command copies from one source device or file `src` to another destination device or file `dest`. Normally you would use this to copy to or from a device or partition, for example to duplicate a filesystem.

If the destination is a device, it must be as large or larger than the source file or device, otherwise the copy will fail. This command cannot do partial copies (see “`copy-device-to-device`”).

*This function is deprecated.* In new code, use the “`copy-device-to-device`” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### **device-index**

```
device-index device
```

This function takes a device name (eg. “`/dev/sdb`”) and returns the index of the device in the list of devices.

Index numbers start from 0. The named device must exist, for example as a string returned from “`list-devices`”.

See also “`list-devices`”, “`part-to-dev`”.

### **df**

```
df
```

This command runs the **df**(1) command to report disk space used.

This command is mostly useful for interactive sessions. It is *not* intended that you try to parse the output string. Use “`statvfs`” from programs.

**df-h**`df-h`

This command runs the `df -h` command to report disk space used in human-readable format.

This command is mostly useful for interactive sessions. It is *not* intended that you try to parse the output string. Use “statvfs” from programs.

**disk-create**

```
disk-create filename format size [backingfile:...] [backingformat:...] [preallocation]
```

Create a blank disk image called *filename* (a host file) with format *format* (usually *raw* or *qcow2*). The size is *size* bytes.

If used with the optional *backingfile* parameter, then a snapshot is created on top of the backing file. In this case, *size* must be passed as *-1*. The size of the snapshot is the same as the size of the backing file, which is discovered automatically. You are encouraged to also pass *backingformat* to describe the format of *backingfile*.

If *filename* refers to a block device, then the device is formatted. The *size* is ignored since block devices have an intrinsic size.

The other optional parameters are:

**preallocation**

If format is *raw*, then this can be either *off* (or *sparse*) or *full* to create a sparse or fully allocated file respectively. The default is *off*.

If format is *qcow2*, then this can be *off* (or *sparse*), *metadata* or *full*. Preallocating metadata can be faster when doing lots of writes, but uses more space. The default is *off*.

**compat**

*qcow2* only: Pass the string *1.1* to use the advanced *qcow2* format supported by *qemu* ≥ 1.1.

**clustersize**

*qcow2* only: Change the *qcow2* cluster size. The default is 65536 (bytes) and this setting may be any power of two between 512 and 2097152.

Note that this call does not add the new disk to the handle. You may need to call “add-drive-opts” separately.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

**disk-format**

```
disk-format filename
```

Detect and return the format of the disk image called *filename*. *filename* can also be a host device, etc. If the format of the image could not be detected, then “unknown” is returned.

Note that detecting the disk format can be insecure under some circumstances. See “CVE-2010-3851” in **guestfs(3)**.

See also: “DISK IMAGE FORMATS” in **guestfs(3)**

**disk-has-backing-file**

```
disk-has-backing-file filename
```

Detect and return whether the disk image *filename* has a backing file.

Note that detecting disk features can be insecure under some circumstances. See “CVE-2010-3851” in **guestfs(3)**.

**disk-virtual-size**

```
disk-virtual-size filename
```

Detect and return the virtual size in bytes of the disk image called *filename*.

Note that detecting disk features can be insecure under some circumstances. See “CVE-2010-3851” in

**guestfs**(3).

## **dmesg**

`dmesg`

This returns the kernel messages (**dmesg**(1) output) from the guest kernel. This is sometimes useful for extended debugging of problems.

Another way to get the same information is to enable verbose messages with “set-verbose” or by setting the environment variable `LIBGUESTFS_DEBUG=1` before running the program.

## **download**

`download remotefilename (filename|-)`

Download file *remotefilename* and save it as *filename* on the local machine.

*filename* can also be a named pipe.

See also “upload”, “cat”.

Use `-` instead of a filename to read/write from stdin/stdout.

## **download-blocks**

`download-blocks device start stop (filename|-) [unallocated:true|false]`

Download the data units from *start* address to *stop* from the disk partition (eg. */dev/sda1*) and save them as *filename* on the local machine.

The use of this API on sparse disk image formats such as QCOW, may result in large zero-filled files downloaded on the host.

The size of a data unit varies across filesystem implementations. On NTFS filesystems data units are referred as clusters while on ExtX ones they are referred as fragments.

If the optional `unallocated` flag is true (default is false), only the unallocated blocks will be extracted. This is useful to detect hidden data or to retrieve deleted files which data units have not been overwritten yet.

Use `-` instead of a filename to read/write from stdin/stdout.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

This command depends on the feature `sleuthkit`. See also “feature-available”.

## **download-inode**

`download-inode device inode (filename|-)`

Download a file given its inode from the disk partition (eg. */dev/sda1*) and save it as *filename* on the local machine.

It is not required to mount the disk to run this command.

The command is capable of downloading deleted or inaccessible files.

Use `-` instead of a filename to read/write from stdin/stdout.

This command depends on the feature `sleuthkit`. See also “feature-available”.

## **download-offset**

`download-offset remotefilename (filename|-) offset size`

Download file *remotefilename* and save it as *filename* on the local machine.

*remotefilename* is read for *size* bytes starting at *offset* (this region must be within the file or device).

Note that there is no limit on the amount of data that can be downloaded with this call, unlike with “pread”, and this call always reads the full amount unless an error occurs.

See also “download”, “pread”.

Use `-` instead of a filename to read/write from stdin/stdout.

**drop-caches**

```
drop-caches whattodrop
```

This instructs the guest kernel to drop its page cache, and/or dentries and inode caches. The parameter `whattodrop` tells the kernel what precisely to drop, see [https://linux-mm.org/Drop\\_Caches](https://linux-mm.org/Drop_Caches)

Setting `whattodrop` to 3 should drop everything.

This automatically calls **sync** (2) before the operation, so that the maximum guest memory is freed.

**du**

```
du path
```

This command runs the `du -s` command to estimate file space usage for `path`.

`path` can be a file or a directory. If `path` is a directory then the estimate includes the contents of the directory and all subdirectories (recursively).

The result is the estimated size in *kilobytes* (ie. units of 1024 bytes).

**e2fsck**

```
e2fsck device [correct:true|false] [forceall:true|false]
```

This runs the ext2/ext3 filesystem checker on `device`. It can take the following optional arguments:

**correct**

Automatically repair the file system. This option will cause `e2fsck` to automatically fix any filesystem problems that can be safely fixed without human intervention.

This option may not be specified at the same time as the `forceall` option.

**forceall**

Assume an answer of ‘yes’ to all questions; allows `e2fsck` to be used non-interactively.

This option may not be specified at the same time as the `correct` option.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

**e2fsck-f**

```
e2fsck-f device
```

This runs `e2fsck -p -f device`, ie. runs the ext2/ext3 filesystem checker on `device`, noninteractively (`-p`), even if the filesystem appears to be clean (`-f`).

*This function is deprecated.* In new code, use the “`e2fsck`” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

**echo-daemon**

```
echo-daemon 'words ...'
```

This command concatenates the list of words passed with single spaces between them and returns the resulting string.

You can use this command to test the connection through to the daemon.

See also “`ping-daemon`”.

**egrep**

```
egrep regex path
```

This calls the external **egrep** (1) program and returns the matching lines.

Because of the message protocol, there is a transfer limit of somewhere between 2MB and 4MB. See “PROTOCOL LIMITS” in **guestfs** (3).

*This function is deprecated.* In new code, use the “`grep`” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that

there are problems with correct use of these functions.

### **egrepi**

`egrepi regex path`

This calls the external `egrep -i` program and returns the matching lines.

Because of the message protocol, there is a transfer limit of somewhere between 2MB and 4MB. See “PROTOCOL LIMITS” in **guestfs**(3).

*This function is deprecated.* In new code, use the “`grep`” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### **equal**

`equal file1 file2`

This compares the two files *file1* and *file2* and returns true if their content is exactly equal, or false otherwise.

The external **cmp**(1) program is used for the comparison.

### **exists**

`exists path`

This returns true if and only if there is a file, directory (or anything) with the given path name.

See also “`is-file`”, “`is-dir`”, “`stat`”.

### **extlinux**

`extlinux directory`

Install the SYSLINUX bootloader on the device mounted at *directory*. Unlike “`syslinux`” which requires a FAT filesystem, this can be used on an ext2/3/4 or btrfs filesystem.

The *directory* parameter can be either a mountpoint, or a directory within the mountpoint.

You also have to mark the partition as “active” (“`part-set-bootable`”) and a Master Boot Record must be installed (eg. using “`pwrite-device`”) on the first sector of the whole disk. The SYSLINUX package comes with some suitable Master Boot Records. See the **extlinux**(1) man page for further information.

Additional configuration can be supplied to SYSLINUX by placing a file called *extlinux.conf* on the filesystem under *directory*. For further information about the contents of this file, see **extlinux**(1).

See also “`syslinux`”.

This command depends on the feature `extlinux`. See also “`feature-available`”.

### **f2fs-expand**

`f2fs-expand device`

This expands a f2fs filesystem to match the size of the underlying device.

This command depends on the feature `f2fs`. See also “`feature-available`”.

### **fallocate**

`fallocate path len`

This command preallocates a file (containing zero bytes) named *path* of size *len* bytes. If the file exists already, it is overwritten.

Do not confuse this with the guestfish-specific `alloc` command which allocates a file in the host and attaches it as a device.

*This function is deprecated.* In new code, use the “`fallocate64`” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

**fallocate64**

```
fallocate64 path len
```

This command preallocates a file (containing zero bytes) named `path` of size `len` bytes. If the file exists already, it is overwritten.

Note that this call allocates disk blocks for the file. To create a sparse file use “truncate-size” instead.

The deprecated call “fallocate” does the same, but owing to an oversight it only allowed 30 bit lengths to be specified, effectively limiting the maximum size of files created through that call to 1GB.

Do not confuse this with the guestfish-specific `alloc` and `sparse` commands which create a file in the host and attach it as a device.

**feature-available**

```
feature-available 'groups ...'
```

This is the same as “available”, but unlike that call it returns a simple true/false boolean result, instead of throwing an exception if a feature is not found. For other documentation see “available”.

**fgrep**

```
fgrep pattern path
```

This calls the external **fgrep**(1) program and returns the matching lines.

Because of the message protocol, there is a transfer limit of somewhere between 2MB and 4MB. See “PROTOCOL LIMITS” in **guestfs**(3).

*This function is deprecated.* In new code, use the “grep” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

**fgrep\_i**

```
fgrep_i pattern path
```

This calls the external `fgrep -i` program and returns the matching lines.

Because of the message protocol, there is a transfer limit of somewhere between 2MB and 4MB. See “PROTOCOL LIMITS” in **guestfs**(3).

*This function is deprecated.* In new code, use the “grep” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

**file**

```
file path
```

This call uses the standard **file**(1) command to determine the type or contents of the file.

This call will also transparently look inside various types of compressed file.

The exact command which runs is `file -zb path`. Note in particular that the filename is not prepended to the output (the `-b` option).

The output depends on the output of the underlying **file**(1) command and it can change in future in ways beyond our control. In other words, the output is not guaranteed by the ABI.

See also: **file**(1), “vfs-type”, “lstat”, “is-file”, “is-blockdev” (etc), “is-zero”.

**file-architecture**

```
file-architecture filename
```

This detects the architecture of the binary *filename*, and returns it if known.

Currently defined architectures are:



“aarch64”

64 bit ARM.

“arm”

32 bit ARM.

“i386”

This string is returned for all 32 bit i386, i486, i586, i686 binaries irrespective of the precise processor requirements of the binary.

“ia64”

Intel Itanium.

“ppc”

32 bit Power PC.

“ppc64”

64 bit Power PC (big endian).

“ppc64le”

64 bit Power PC (little endian).

“riscv32”

“riscv64”

“riscv128”

RISC-V 32-, 64- or 128-bit variants.

“s390”

31 bit IBM S/390.

“s390x”

64 bit IBM S/390.

“sparc”

32 bit SPARC.

“sparc64”

64 bit SPARC V9 and above.

“x86\_64”

64 bit x86-64.

Libguestfs may return other architecture strings in future.

The function works on at least the following types of files:

- many types of Un\*x and Linux binary
- many types of Un\*x and Linux shared library
- Windows Win32 and Win64 binaries
- Windows Win32 and Win64 DLLs
- Win32 binaries and DLLs return i386.
- Win64 binaries and DLLs return x86\_64.
- Linux kernel modules
- Linux new-style initrd images
- some non-x86 Linux vmlinuz kernels

What it can't do currently:

- static libraries (libfoo.a)
- Linux old-style initrd as compressed ext2 filesystem (RHEL 3)

- x86 Linux vmlinuz kernels

x86 vmlinuz images (bzImage format) consist of a mix of 16-, 32- and compressed code, and are horribly hard to unpack. If you want to find the architecture of a kernel, use the architecture of the associated initrd or kernel module(s) instead.

### filesize

```
filesize file
```

This command returns the size of *file* in bytes.

To get other stats about a file, use “stat”, “lstat”, “is-dir”, “is-file” etc. To get the size of block devices, use “blockdev-getsize64”.

### filesystem-available

```
filesystem-available filesystem
```

Check whether libguestfs supports the named filesystem. The argument *filesystem* is a filesystem name, such as ext3.

You must call “launch” before using this command.

This is mainly useful as a negative test. If this returns true, it doesn’t mean that a particular filesystem can be created or mounted, since filesystems can fail for other reasons such as it being a later version of the filesystem, or having incompatible features, or lacking the right mkfs.<*fs*> tool.

See also “available”, “feature-available”, “AVAILABILITY” in **guestfs** (3).

### filesystem-walk

```
filesystem-walk device
```

Walk through the internal structures of a disk partition (eg. */dev/sda1*) in order to return a list of all the files and directories stored within.

It is not necessary to mount the disk partition to run this command.

All entries in the filesystem are returned. This function can list deleted or unaccessible files. The entries are *not* sorted.

The `tsk_dirent` structure contains the following fields.

`tsk_inode`

Filesystem reference number of the node. It might be 0 if the node has been deleted.

`tsk_type`

Basic file type information. See below for a detailed list of values.

`tsk_size`

File size in bytes. It might be -1 if the node has been deleted.

`tsk_name`

The file path relative to its directory.

`tsk_flags`

Bitfield containing extra information regarding the entry. It contains the logical OR of the following values:

0x0001

If set to 1, the file is allocated and visible within the filesystem. Otherwise, the file has been deleted. Under certain circumstances, the function `download_inode` can be used to recover deleted files.

0x0002

Filesystem such as NTFS and Ext2 or greater, separate the file name from the metadata structure. The bit is set to 1 when the file name is in an unallocated state and the metadata structure is in an allocated one. This generally implies the metadata has been reallocated to a new file. Therefore, information such as file type, file size, timestamps, number of links and symlink target might not

correspond with the ones of the original deleted entry.

0x0004

The bit is set to 1 when the file is compressed using filesystem native compression support (NTFS). The API is not able to detect application level compression.

tsk\_atime\_sec  
tsk\_atime\_nsec  
tsk\_mtime\_sec  
tsk\_mtime\_nsec  
tsk\_ctime\_sec  
tsk\_ctime\_nsec  
tsk\_crtime\_sec  
tsk\_crtime\_nsec

Respectively, access, modification, last status change and creation time in Unix format in seconds and nanoseconds.

tsk\_nlink

Number of file names pointing to this entry.

tsk\_link

If the entry is a symbolic link, this field will contain the path to the target file.

The `tsk_type` field will contain one of the following characters:

'b' Block special  
'c' Char special  
'd' Directory  
'f' FIFO (named pipe)  
'l' Symbolic link  
'r' Regular file  
's' Socket  
'h' Shadow inode (Solaris)  
'w' Whiteout inode (BSD)  
'u' Unknown file type

This command depends on the feature `libtsk`. See also “feature-available”.

### **fill**

`fill c len path`

This command creates a new file called `path`. The initial content of the file is `len` octets of `c`, where `c` must be a number in the range `[ 0 . . 255 ]`.

To fill a file with zero bytes (sparsely), it is much more efficient to use “truncate-size”. To create a file with a pattern of repeating bytes use “fill-pattern”.

### **fill-dir**

`fill-dir dir nr`

This function, useful for testing filesystems, creates `nr` empty files in the directory `dir` with names 00000000 through `nr-1` (ie. each file name is 8 digits long padded with zeroes).

### **fill-pattern**

`fill-pattern pattern len path`

This function is like “fill” except that it creates a new file of length `len` containing the repeating pattern of bytes in `pattern`. The pattern is truncated if necessary to ensure the length of the file is exactly `len` bytes.

**find**

```
find directory
```

This command lists out all files and directories, recursively, starting at *directory*. It is essentially equivalent to running the shell command `find directory -print` but some post-processing happens on the output, described below.

This returns a list of strings *without any prefix*. Thus if the directory structure was:

```
/tmp/a
/tmp/b
/tmp/c/d
```

then the returned list from “find” */tmp* would be 4 elements:

```
a
b
c
c/d
```

If *directory* is not a directory, then this command returns an error.

The returned list is sorted.

**find0**

```
find0 directory (files|-)
```

This command lists out all files and directories, recursively, starting at *directory*, placing the resulting list in the external file called *files*.

This command works the same way as “find” with the following exceptions:

- The resulting list is written to an external file.
- Items (filenames) in the result are separated by `\0` characters. See **find** (1) option `-print0`.
- The result list is not sorted.

Use `-` instead of a filename to read/write from stdin/stdout.

**find-inode**

```
find-inode device inode
```

Searches all the entries associated with the given inode.

For each entry, a `tsk_dirent` structure is returned. See `filesystem_walk` for more information about `tsk_dirent` structures.

This command depends on the feature `libtsk`. See also “feature-available”.

**findfs-label**

```
findfs-label label
```

This command searches the filesystems and returns the one which has the given label. An error is returned if no such filesystem can be found.

To find the label of a filesystem, use “`vfs-label`”.

**findfs-uuid**

```
findfs-uuid uuid
```

This command searches the filesystems and returns the one which has the given UUID. An error is returned if no such filesystem can be found.

To find the UUID of a filesystem, use “`vfs-uuid`”.

**fsck**

```
fsck fstype device
```

This runs the filesystem checker (fsck) on device which should have filesystem type *fstype*.

The returned integer is the status. See **fsck** (8) for the list of status codes from **fsck**.

Notes:

- Multiple status codes can be summed together.
- A non-zero return code can mean “success”, for example if errors have been corrected on the filesystem.
- Checking or repairing NTFS volumes is not supported (by linux-ntfs).

This command is entirely equivalent to running **fsck -a -t fstype device**.

### **fstrim**

```
fstrim mountpoint [offset:N] [length:N] [minimumfreeextent:N]
```

Trim the free space in the filesystem mounted on **mountpoint**. The filesystem must be mounted read-write.

The filesystem contents are not affected, but any free space in the filesystem is “trimmed”, that is, given back to the host device, thus making disk images more sparse, allowing unused space in qcow2 files to be reused, etc.

This operation requires support in libguestfs, the mounted filesystem, the host filesystem, qemu and the host kernel. If this support isn’t present it may give an error or even appear to run but do nothing.

In the case where the kernel vfs driver does not support trimming, this call will fail with **errno** set to **ENOTSUP**. Currently this happens when trying to trim FAT filesystems.

See also “zero-free-space”. That is a slightly different operation that turns free space in the filesystem into zeroes. It is valid to call “fstrim” either instead of, or after calling “zero-free-space”.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

This command depends on the feature **fstrim**. See also “feature-available”.

### **get-append**

```
get-append
```

Return the additional kernel options which are added to the libguestfs appliance kernel command line.

If **NULL** then no options are added.

### **get-attach-method**

```
get-attach-method
```

Return the current backend.

See “set-backend” and “BACKEND” in **guestfs** (3).

*This function is deprecated.* In new code, use the “get-backend” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### **get-autosync**

```
get-autosync
```

Get the autosync flag.

### **get-backend**

```
get-backend
```

Return the current backend.

This handle property was previously called the “attach method”.

See “set-backend” and “BACKEND” in **guestfs** (3).

**get-backend-setting**

get-backend-setting name

Find a backend setting string which is either "name" or begins with "name=". If "name", this returns the string "1". If "name=", this returns the part after the equals sign (which may be an empty string).

If no such setting is found, this function throws an error. The errno (see “last-errno”) will be ESRCH in this case.

See “BACKEND” in **guestfs**(3), “BACKEND SETTINGS” in **guestfs**(3).

**get-backend-settings**

get-backend-settings

Return the current backend settings.

This call returns all backend settings strings. If you want to find a single backend setting, see “get-backend-setting”.

See “BACKEND” in **guestfs**(3), “BACKEND SETTINGS” in **guestfs**(3).

**get-cachedir**

get-cachedir

Get the directory used by the handle to store the appliance cache.

**get-direct**

get-direct

Return the direct appliance mode flag.

*This function is deprecated.* In new code, use the “internal-get-console-socket” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

**get-e2attrs**

get-e2attrs file

This returns the file attributes associated with *file*.

The attributes are a set of bits associated with each inode which affect the behaviour of the file. The attributes are returned as a string of letters (described below). The string may be empty, indicating that no file attributes are set for this file.

These attributes are only present when the file is located on an ext2/3/4 filesystem. Using this call on other filesystem types will result in an error.

The characters (file attributes) in the returned string are currently:

'A' When the file is accessed, its atime is not modified.

'a' The file is append-only.

'c' The file is compressed on-disk.

'D' (Directories only.) Changes to this directory are written synchronously to disk.

'd' The file is not a candidate for backup (see **dump**(8)).

'E' The file has compression errors.

'e' The file is using extents.

'h' The file is storing its blocks in units of the filesystem blocksize instead of sectors.

'I' (Directories only.) The directory is using hashed trees.

'i' The file is immutable. It cannot be modified, deleted or renamed. No link can be created to this file.

'j' The file is data-journalled.

- 's' When the file is deleted, all its blocks will be zeroed.
- 'S' Changes to this file are written synchronously to disk.
- 'T' (Directories only.) This is a hint to the block allocator that subdirectories contained in this directory should be spread across blocks. If not present, the block allocator will try to group subdirectories together.
- 't' For a file, this disables tail-merging. (Not used by upstream implementations of ext2.)
- 'u' When the file is deleted, its blocks will be saved, allowing the file to be undeleted.
- 'X' The raw contents of the compressed file may be accessed.
- 'Z' The compressed file is dirty.

More file attributes may be added to this list later. Not all file attributes may be set for all kinds of files. For detailed information, consult the **chattr**(1) man page.

See also “set-e2attrs”.

Don't confuse these attributes with extended attributes (see “getxattr”).

### **get-e2generation**

`get-e2generation file`

This returns the ext2 file generation of a file. The generation (which used to be called the “version”) is a number associated with an inode. This is most commonly used by NFS servers.

The generation is only present when the file is located on an ext2/3/4 filesystem. Using this call on other filesystem types will result in an error.

See “set-e2generation”.

### **get-e2label**

`get-e2label device`

This returns the ext2/3/4 filesystem label of the filesystem on device.

*This function is deprecated.* In new code, use the “vfs-label” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### **get-e2uuid**

`get-e2uuid device`

This returns the ext2/3/4 filesystem UUID of the filesystem on device.

*This function is deprecated.* In new code, use the “vfs-uuid” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### **get-hv**

`get-hv`

Return the current hypervisor binary.

This is always non-NULL. If it wasn't set already, then this will return the default qemu binary name.

### **get-identifier**

`get-identifier`

Get the handle identifier. See “set-identifier”.

### **get-libvirt-requested-credential-challenge**

`get-libvirt-requested-credential-challenge index`

Get the challenge (provided by libvirt) for the `index`'th requested credential. If libvirt did not provide a challenge, this returns the empty string “”.

See “LIBVIRT AUTHENTICATION” in **guestfs**(3) for documentation and example code.

### **get-libvirt-requested-credential-defresult**

`get-libvirt-requested-credential-defresult index`

Get the default result (provided by libvirt) for the `index`’th requested credential. If libvirt did not provide a default result, this returns the empty string `" "`.

See “LIBVIRT AUTHENTICATION” in **guestfs**(3) for documentation and example code.

### **get-libvirt-requested-credential-prompt**

`get-libvirt-requested-credential-prompt index`

Get the prompt (provided by libvirt) for the `index`’th requested credential. If libvirt did not provide a prompt, this returns the empty string `" "`.

See “LIBVIRT AUTHENTICATION” in **guestfs**(3) for documentation and example code.

### **get-libvirt-requested-credentials**

`get-libvirt-requested-credentials`

This should only be called during the event callback for events of type `GUESTFS_EVENT_LIBVIRT_AUTH`.

Return the list of credentials requested by libvirt. Possible values are a subset of the strings provided when you called “set-libvirt-supported-credentials”.

See “LIBVIRT AUTHENTICATION” in **guestfs**(3) for documentation and example code.

### **get-memsize**

`get-memsize`

This gets the memory size in megabytes allocated to the hypervisor.

If “set-memsize” was not called on this handle, and if `LIBGUESTFS_MEMSIZE` was not set, then this returns the compiled-in default value for memsize.

For more information on the architecture of libguestfs, see **guestfs**(3).

### **get-network**

`get-network`

This returns the enable network flag.

### **get-path**

`get-path`

Return the current search path.

This is always non-NULL. If it wasn’t set already, then this will return the default path.

### **get-pgroup**

`get-pgroup`

This returns the process group flag.

### **get-pid**

#### **pid**

`get-pid`

Return the process ID of the hypervisor. If there is no hypervisor running, then this will return an error.

This is an internal call used for debugging and testing.

### **get-program**

`get-program`

Get the program name. See “set-program”.



**get-qemu**`get-qemu`

Return the current hypervisor binary (usually qemu).

This is always non-NULL. If it wasn't set already, then this will return the default qemu binary name.

*This function is deprecated.* In new code, use the “get-hv” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

**get-recovery-proc**`get-recovery-proc`

Return the recovery process enabled flag.

**get-selinux**`get-selinux`

This returns the current setting of the selinux flag which is passed to the appliance at boot time. See “set-selinux”.

For more information on the architecture of libguestfs, see **guestfs**(3).

*This function is deprecated.* In new code, use the “selinux-relabel” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

**get-smp**`get-smp`

This returns the number of virtual CPUs assigned to the appliance.

**get-sockdir**`get-sockdir`

Get the directory used by the handle to store temporary socket files.

This is different from “get-tmpdir”, as we need shorter paths for sockets (due to the limited buffers of filenames for UNIX sockets), and “get-tmpdir” may be too long for them.

The environment variable `XDG_RUNTIME_DIR` controls the default value: If `XDG_RUNTIME_DIR` is set, then that is the default. Else `/tmp` is the default.

**get-tmpdir**`get-tmpdir`

Get the directory used by the handle to store temporary files.

**get-trace**`get-trace`

Return the command trace flag.

**get-umask**`get-umask`

Return the current umask. By default the umask is 022 unless it has been set by calling “umask”.

**get-verbose**`get-verbose`

This returns the verbose messages flag.

**getcon**`getcon`

This gets the SELinux security context of the daemon.

See the documentation about SELINUX in **guestfs**(3), and “setcon”

*This function is deprecated.* In new code, use the “selinux-relabel” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

This command depends on the feature `selinux`. See also “feature-available”.

### **getxattr**

```
getxattr path name
```

Get a single extended attribute from file `path` named `name`. This call follows symlinks. If you want to lookup an extended attribute for the symlink itself, use “lgetxattr”.

Normally it is better to get all extended attributes from a file in one go by calling “getxattrs”. However some Linux filesystem implementations are buggy and do not provide a way to list out attributes. For these filesystems (notably ntfs-3g) you have to know the names of the extended attributes you want in advance and call this function.

Extended attribute values are blobs of binary data. If there is no extended attribute named `name`, this returns an error.

See also: “getxattrs”, “lgetxattr”, **attr**(5).

This command depends on the feature `linuxxattrs`. See also “feature-available”.

### **getxattrs**

```
getxattrs path
```

This call lists the extended attributes of the file or directory `path`.

At the system call level, this is a combination of the **listxattr**(2) and **getxattr**(2) calls.

See also: “lgetxattrs”, **attr**(5).

This command depends on the feature `linuxxattrs`. See also “feature-available”.

### **glob-expand**

#### **glob-expand-opts**

```
glob-expand pattern [directoryslash:true|false]
```

This command searches for all the pathnames matching `pattern` according to the wildcard expansion rules used by the shell.

If no paths match, then this returns an empty list (note: not an error).

It is just a wrapper around the C **glob**(3) function with flags `GLOB_MARK|GLOB_BRACE`. See that manual page for more details.

`directoryslash` controls whether use the `GLOB_MARK` flag for **glob**(3), and it defaults to true. It can be explicitly set as off to return no trailing slashes in filenames of directories.

Notice that there is no equivalent command for expanding a device name (eg. `/dev/sd*`). Use “list-devices”, “list-partitions” etc functions instead.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

### **grep**

#### **grep-opts**

```
grep regex path [extended:true|false] [fixed:true|false] [insensitive:true|false]
```

This calls the external **grep**(1) program and returns the matching lines.

The optional flags are:

`extended`

Use extended regular expressions. This is the same as using the `-E` flag.

**fixed**

Match fixed (don't use regular expressions). This is the same as using the `-F` flag.

**insensitive**

Match case-insensitive. This is the same as using the `-i` flag.

**compressed**

Use **zgrep**(1) instead of **grep**(1). This allows the input to be compress- or gzip-compressed.

This command has one or more optional arguments. See "OPTIONAL ARGUMENTS".

Because of the message protocol, there is a transfer limit of somewhere between 2MB and 4MB. See "PROTOCOL LIMITS" in **guestfs**(3).

## **grepi**

`grepi regex path`

This calls the external `grep -i` program and returns the matching lines.

Because of the message protocol, there is a transfer limit of somewhere between 2MB and 4MB. See "PROTOCOL LIMITS" in **guestfs**(3).

*This function is deprecated.* In new code, use the "grep" call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

## **grub-install**

`grub-install root device`

This command installs GRUB 1 (the Grand Unified Bootloader) on `device`, with the root directory being `root`.

Notes:

- There is currently no way in the API to install grub2, which is used by most modern Linux guests. It is possible to run the `grub2` command from the guest, although see the caveats in "RUNNING COMMANDS" in **guestfs**(3).
- This uses **grub-install**(8) from the host. Unfortunately grub is not always compatible with itself, so this only works in rather narrow circumstances. Careful testing with each guest version is advisable.
- If `grub-install` reports the error "No suitable drive was found in the generated device map." it may be that you need to create a `/boot/grub/device.map` file first that contains the mapping between grub device names and Linux device names. It is usually sufficient to create a file containing:

```
(hd0) /dev/vda
```

replacing `/dev/vda` with the name of the installation device.

This command depends on the feature `grub`. See also "feature-available".

## **head**

`head path`

This command returns up to the first 10 lines of a file as a list of strings.

Because of the message protocol, there is a transfer limit of somewhere between 2MB and 4MB. See "PROTOCOL LIMITS" in **guestfs**(3).

## **head-n**

`head-n nrlines path`

If the parameter `nrlines` is a positive number, this returns the first `nrlines` lines of the file `path`.

If the parameter `nrlines` is a negative number, this returns lines from the file `path`, excluding the last `nrlines` lines.

If the parameter `nrlines` is zero, this returns an empty list.

Because of the message protocol, there is a transfer limit of somewhere between 2MB and 4MB. See “PROTOCOL LIMITS” in **guestfs**(3).

**hexdump**

`hexdump path`

This runs `hexdump -C` on the given `path`. The result is the human-readable, canonical hex dump of the file.

Because of the message protocol, there is a transfer limit of somewhere between 2MB and 4MB. See “PROTOCOL LIMITS” in **guestfs**(3).

**hivex-close**

`hivex-close`

Close the current hivex handle.

This is a wrapper around the **hivex**(3) call of the same name.

This command depends on the feature `hivex`. See also “feature-available”.

**hivex-commit**

`hivex-commit filename`

Commit (write) changes to the hive.

If the optional *filename* parameter is null, then the changes are written back to the same hive that was opened. If this is not null then they are written to the alternate filename given and the original hive is left untouched.

This is a wrapper around the **hivex**(3) call of the same name.

This command depends on the feature `hivex`. See also “feature-available”.

**hivex-node-add-child**

`hivex-node-add-child parent name`

Add a child node to parent named `name`.

This is a wrapper around the **hivex**(3) call of the same name.

This command depends on the feature `hivex`. See also “feature-available”.

**hivex-node-children**

`hivex-node-children nodeh`

Return the list of nodes which are subkeys of `nodeh`.

This is a wrapper around the **hivex**(3) call of the same name.

This command depends on the feature `hivex`. See also “feature-available”.

**hivex-node-delete-child**

`hivex-node-delete-child nodeh`

Delete `nodeh`, recursively if necessary.

This is a wrapper around the **hivex**(3) call of the same name.

This command depends on the feature `hivex`. See also “feature-available”.

**hivex-node-get-child**

`hivex-node-get-child nodeh name`

Return the child of `nodeh` with the name `name`, if it exists. This can return 0 meaning the name was not found.

This is a wrapper around the **hivex**(3) call of the same name.

This command depends on the feature `hivex`. See also “feature-available”.

**hivex-node-get-value**

```
hivex-node-get-value nodeh key
```

Return the value attached to *nodeh* which has the name *key*, if it exists. This can return 0 meaning the key was not found.

This is a wrapper around the **hivex** (3) call of the same name.

This command depends on the feature *hivex*. See also “feature-available”.

**hivex-node-name**

```
hivex-node-name nodeh
```

Return the name of *nodeh*.

This is a wrapper around the **hivex** (3) call of the same name.

This command depends on the feature *hivex*. See also “feature-available”.

**hivex-node-parent**

```
hivex-node-parent nodeh
```

Return the parent node of *nodeh*.

This is a wrapper around the **hivex** (3) call of the same name.

This command depends on the feature *hivex*. See also “feature-available”.

**hivex-node-set-value**

```
hivex-node-set-value nodeh key t val
```

Set or replace a single value under the node *nodeh*. The *key* is the name, *t* is the type, and *val* is the data.

This is a wrapper around the **hivex** (3) call of the same name.

This command depends on the feature *hivex*. See also “feature-available”.

**hivex-node-values**

```
hivex-node-values nodeh
```

Return the array of (key, datatype, data) tuples attached to *nodeh*.

This is a wrapper around the **hivex** (3) call of the same name.

This command depends on the feature *hivex*. See also “feature-available”.

**hivex-open**

```
hivex-open filename [verbose:true|false] [debug:true|false] [write:true|false] [...]
```

Open the Windows Registry hive file named *filename*. If there was any previous *hivex* handle associated with this *guestfs* session, then it is closed.

This is a wrapper around the **hivex** (3) call of the same name.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

This command depends on the feature *hivex*. See also “feature-available”.

**hivex-root**

```
hivex-root
```

Return the root node of the hive.

This is a wrapper around the **hivex** (3) call of the same name.

This command depends on the feature *hivex*. See also “feature-available”.

**hivex-value-key**

```
hivex-value-key valueh
```

Return the key (name) field of a (key, datatype, data) tuple.

This is a wrapper around the **hivex** (3) call of the same name.

This command depends on the feature **hivex**. See also “feature-available”.

#### **hivex-value-string**

```
hivex-value-string valueh
```

This calls “hivex-value-value” (which returns the data field from a hivex value tuple). It then assumes that the field is a UTF-16LE string and converts the result to UTF-8 (or if this is not possible, it returns an error).

This is useful for reading strings out of the Windows registry. However it is not foolproof because the registry is not strongly-typed and fields can contain arbitrary or unexpected data.

This command depends on the feature **hivex**. See also “feature-available”.

#### **hivex-value-type**

```
hivex-value-type valueh
```

Return the data type field from a (key, datatype, data) tuple.

This is a wrapper around the **hivex** (3) call of the same name.

This command depends on the feature **hivex**. See also “feature-available”.

#### **hivex-value-utf8**

```
hivex-value-utf8 valueh
```

This calls “hivex-value-value” (which returns the data field from a hivex value tuple). It then assumes that the field is a UTF-16LE string and converts the result to UTF-8 (or if this is not possible, it returns an error).

This is useful for reading strings out of the Windows registry. However it is not foolproof because the registry is not strongly-typed and fields can contain arbitrary or unexpected data.

*This function is deprecated.* In new code, use the “hivex-value-string” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

This command depends on the feature **hivex**. See also “feature-available”.

#### **hivex-value-value**

```
hivex-value-value valueh
```

Return the data field of a (key, datatype, data) tuple.

This is a wrapper around the **hivex** (3) call of the same name.

See also: “hivex-value-utf8”.

This command depends on the feature **hivex**. See also “feature-available”.

#### **initrd-cat**

```
initrd-cat initrdpath filename
```

This command unpacks the file *filename* from the initrd file called *initrdpath*. The filename must be given *without* the initial / character.

For example, in guestfish you could use the following command to examine the boot script (usually called */init*) contained in a Linux initrd or initramfs image:

```
initrd-cat /boot/initrd-<version>.img init
```

See also “initrd-list”.

Because of the message protocol, there is a transfer limit of somewhere between 2MB and 4MB. See “PROTOCOL LIMITS” in **guestfs** (3).

#### **initrd-list**

```
initrd-list path
```

This command lists out files contained in an initrd.

The files are listed without any initial / character. The files are listed in the order they appear (not necessarily alphabetical). Directory names are listed as separate items.

Old Linux kernels (2.4 and earlier) used a compressed ext2 filesystem as initrd. We *only* support the newer initramfs format (compressed cpio files).

### **inotify-add-watch**

```
inotify-add-watch path mask
```

Watch path for the events listed in mask.

Note that if path is a directory then events within that directory are watched, but this does *not* happen recursively (in subdirectories).

Note for non-C or non-Linux callers: the inotify events are defined by the Linux kernel ABI and are listed in */usr/include/sys/inotify.h*.

This command depends on the feature `inotify`. See also “feature-available”.

### **inotify-close**

```
inotify-close
```

This closes the inotify handle which was previously opened by `inotify_init`. It removes all watches, throws away any pending events, and deallocates all resources.

This command depends on the feature `inotify`. See also “feature-available”.

### **inotify-files**

```
inotify-files
```

This function is a helpful wrapper around “`inotify-read`” which just returns a list of pathnames of objects that were touched. The returned pathnames are sorted and deduplicated.

This command depends on the feature `inotify`. See also “feature-available”.

### **inotify-init**

```
inotify-init maxevents
```

This command creates a new inotify handle. The inotify subsystem can be used to notify events which happen to objects in the guest filesystem.

`maxevents` is the maximum number of events which will be queued up between calls to “`inotify-read`” or “`inotify-files`”. If this is passed as 0, then the kernel (or previously set) default is used. For Linux 2.6.29 the default was 16384 events. Beyond this limit, the kernel throws away events, but records the fact that it threw them away by setting a flag `IN_Q_OVERFLOW` in the returned structure list (see “`inotify-read`”).

Before any events are generated, you have to add some watches to the internal watch list. See: “`inotify-add-watch`” and “`inotify-rm-watch`”.

Queued up events should be read periodically by calling “`inotify-read`” (or “`inotify-files`” which is just a helpful wrapper around “`inotify-read`”). If you don’t read the events out often enough then you risk the internal queue overflowing.

The handle should be closed after use by calling “`inotify-close`”. This also removes any watches automatically.

See also **inotify**(7) for an overview of the inotify interface as exposed by the Linux kernel, which is roughly what we expose via `libguestfs`. Note that there is one global inotify handle per `libguestfs` instance.

This command depends on the feature `inotify`. See also “feature-available”.

### **inotify-read**

```
inotify-read
```

Return the complete queue of events that have happened since the previous read call.

If no events have happened, this returns an empty list.

*Note:* In order to make sure that all events have been read, you must call this function repeatedly until it returns an empty list. The reason is that the call will read events up to the maximum appliance-to-host message size and leave remaining events in the queue.

This command depends on the feature `inotify`. See also “feature-available”.

### **inotify-rm-watch**

```
inotify-rm-watch wd
```

Remove a previously defined inotify watch. See “inotify-add-watch”.

This command depends on the feature `inotify`. See also “feature-available”.

### **inspect-get-arch**

```
inspect-get-arch root
```

This returns the architecture of the inspected operating system. The possible return values are listed under “file-architecture”.

If the architecture could not be determined, then the string `unknown` is returned.

Please read “INSPECTION” in **guestfs**(3) for more details.

### **inspect-get-distro**

```
inspect-get-distro root
```

This returns the distro (distribution) of the inspected operating system.

Currently defined distros are:

“alpinelinux”

Alpine Linux.

“altlinux”

ALT Linux.

“archlinux”

Arch Linux.

“buildroot”

Buildroot-derived distro, but not one we specifically recognize.

“centos”

CentOS.

“cirros”

Cirros.

“coreos”

CoreOS.

“debian”

Debian.

“fedora”

Fedora.

“freebsd”

FreeBSD.

“freedos”

FreeDOS.

“frugalware”

Frugalware.

“gentoo”

Gentoo.



“kalilinux”  
Kali Linux.

“linuxmint”  
Linux Mint.

“mageia”  
Mageia.

“mandriva”  
Mandriva.

“meego”  
MeeGo.

“msdos”  
Microsoft DOS.

“neokylin”  
NeoKylin.

“netbsd”  
NetBSD.

“openbsd”  
OpenBSD.

“openmandriva”  
OpenMandriva Lx.

“opensuse”  
OpenSUSE.

“oraclelinux”  
Oracle Linux.

“pardus”  
Pardus.

“pldlinux”  
PLD Linux.

“redhat-based”  
Some Red Hat-derived distro.

“rhel”  
Red Hat Enterprise Linux.

“scientificlinux”  
Scientific Linux.

“slackware”  
Slackware.

“sles”  
SuSE Linux Enterprise Server or Desktop.

“suse-based”  
Some openSuSE-derived distro.

“ttylinux”  
ttylinux.

“ubuntu”  
Ubuntu.

“unknown”

The distro could not be determined.

“voidlinux”

Void Linux.

“windows”

Windows does not have distributions. This string is returned if the OS type is Windows.

Future versions of libguestfs may return other strings here. The caller should be prepared to handle any string.

Please read “INSPECTION” in **guestfs**(3) for more details.

### **inspect-get-drive-mappings**

```
inspect-get-drive-mappings root
```

This call is useful for Windows which uses a primitive system of assigning drive letters (like C:\) to partitions. This inspection API examines the Windows Registry to find out how disks/partitions are mapped to drive letters, and returns a hash table as in the example below:

```
C      =>    /dev/vda2
E      =>    /dev/vdb1
F      =>    /dev/vdc1
```

Note that keys are drive letters. For Windows, the key is case insensitive and just contains the drive letter, without the customary colon separator character.

In future we may support other operating systems that also used drive letters, but the keys for those might not be case insensitive and might be longer than 1 character. For example in OS-9, hard drives were named h0, h1 etc.

For Windows guests, currently only hard drive mappings are returned. Removable disks (eg. DVD-ROMs) are ignored.

For guests that do not use drive mappings, or if the drive mappings could not be determined, this returns an empty hash table.

Please read “INSPECTION” in **guestfs**(3) for more details. See also “inspect-get-mountpoints”, “inspect-get-filesystems”.

### **inspect-get-filesystems**

```
inspect-get-filesystems root
```

This returns a list of all the filesystems that we think are associated with this operating system. This includes the root filesystem, other ordinary filesystems, and non-mounted devices like swap partitions.

In the case of a multi-boot virtual machine, it is possible for a filesystem to be shared between operating systems.

Please read “INSPECTION” in **guestfs**(3) for more details. See also “inspect-get-mountpoints”.

### **inspect-get-format**

```
inspect-get-format root
```

Before libguestfs 1.38, there was some unreliable support for detecting installer CDs. This API would return:

```
installed
```

This is an installed operating system.

```
installer
```

The disk image being inspected is not an installed operating system, but a *bootable* install disk, live CD, or similar.

unknown

The format of this disk image is not known.

In libguestfs  $\geq 1.38$ , this only returns `installed`. Use `libosinfo` directly to detect installer CDs.

Please read “INSPECTION” in **guestfs**(3) for more details.

*This function is deprecated.* There is no replacement. Consult the API documentation in **guestfs**(3) for further information.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### inspect-get-hostname

```
inspect-get-hostname root
```

This function returns the hostname of the operating system as found by inspection of the guest’s configuration files.

If the hostname could not be determined, then the string `unknown` is returned.

Please read “INSPECTION” in **guestfs**(3) for more details.

### inspect-get-icon

```
inspect-get-icon root [favicon:true|false] [highquality:true|false]
```

This function returns an icon corresponding to the inspected operating system. The icon is returned as a buffer containing a PNG image (re-encoded to PNG if necessary).

If it was not possible to get an icon this function returns a zero-length (non-NULL) buffer. *Callers must check for this case.*

Libguestfs will start by looking for a file called `/etc/favicon.png` or `C:\etc\favicon.png` and if it has the correct format, the contents of this file will be returned. You can disable favicons by passing the optional `favicon` boolean as `false` (default is `true`).

If finding the favicon fails, then we look in other places in the guest for a suitable icon.

If the optional `highquality` boolean is `true` then only high quality icons are returned, which means only icons of high resolution with an alpha channel. The default (`false`) is to return any icon we can, even if it is of substandard quality.

Notes:

- Unlike most other inspection API calls, the guest’s disks must be mounted up before you call this, since it needs to read information from the guest filesystem during the call.
- **Security:** The icon data comes from the untrusted guest, and should be treated with caution. PNG files have been known to contain exploits. Ensure that `libpng` (or other relevant libraries) are fully up to date before trying to process or display the icon.
- The PNG image returned can be any size. It might not be square. Libguestfs tries to return the largest, highest quality icon available. The application must scale the icon to the required size.
- Extracting icons from Windows guests requires the external **wrestool**(1) program from the `icoutils` package, and several programs (**bmptopnm**(1), **pnmtopng**(1), **pamcut**(1)) from the `netpbm` package. These must be installed separately.
- Operating system icons are usually trademarks. Seek legal advice before using trademarks in applications.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

### inspect-get-major-version

```
inspect-get-major-version root
```

This returns the major version number of the inspected operating system.

Windows uses a consistent versioning scheme which is *not* reflected in the popular public names used by

the operating system. Notably the operating system known as “Windows 7” is really version 6.1 (ie. major = 6, minor = 1). You can find out the real versions corresponding to releases of Windows by consulting Wikipedia or MSDN.

If the version could not be determined, then 0 is returned.

Please read “INSPECTION” in **guestfs** (3) for more details.

### **inspect-get-minor-version**

```
inspect-get-minor-version root
```

This returns the minor version number of the inspected operating system.

If the version could not be determined, then 0 is returned.

Please read “INSPECTION” in **guestfs** (3) for more details. See also “inspect-get-major-version”.

### **inspect-get-mountpoints**

```
inspect-get-mountpoints root
```

This returns a hash of where we think the filesystems associated with this operating system should be mounted. Callers should note that this is at best an educated guess made by reading configuration files such as */etc/fstab*. *In particular note* that this may return filesystems which are non-existent or not mountable and callers should be prepared to handle or ignore failures if they try to mount them.

Each element in the returned hashtable has a key which is the path of the mountpoint (eg. */boot*) and a value which is the filesystem that would be mounted there (eg. */dev/sda1*).

Non-mounted devices such as swap devices are *not* returned in this list.

For operating systems like Windows which still use drive letters, this call will only return an entry for the first drive “mounted on” /. For information about the mapping of drive letters to partitions, see “inspect-get-drive-mappings”.

Please read “INSPECTION” in **guestfs** (3) for more details. See also “inspect-get-filesystems”.

### **inspect-get-osinfo**

```
inspect-get-osinfo root
```

This function returns a possible short ID for libosinfo corresponding to the guest.

*Note:* The returned ID is only a guess by libguestfs, and nothing ensures that it actually exists in osinfo-db.

If no ID could not be determined, then the string unknown is returned.

### **inspect-get-package-format**

```
inspect-get-package-format root
```

This function and “inspect-get-package-management” return the package format and package management tool used by the inspected operating system. For example for Fedora these functions would return *rpm* (package format), and *yum* or *dnf* (package management).

This returns the string unknown if we could not determine the package format *or* if the operating system does not have a real packaging system (eg. Windows).

Possible strings include: *rpm*, *deb*, *ebuild*, *pisi*, *pacman*, *pkgsrc*, *apk*, *xbps*. Future versions of libguestfs may return other strings.

Please read “INSPECTION” in **guestfs** (3) for more details.

### **inspect-get-package-management**

```
inspect-get-package-management root
```

“inspect-get-package-format” and this function return the package format and package management tool used by the inspected operating system. For example for Fedora these functions would return *rpm* (package format), and *yum* or *dnf* (package management).

This returns the string unknown if we could not determine the package management tool *or* if the operating system does not have a real packaging system (eg. Windows).

Possible strings include: `yum`, `dnf`, `up2date`, `apt` (for all Debian derivatives), `portage`, `pkg`, `pacman`, `urpmi`, `zypper`, `apk`, `xbps`. Future versions of `libguestfs` may return other strings.

Please read “INSPECTION” in `guestfs` (3) for more details.

### **inspect-get-product-name**

```
inspect-get-product-name root
```

This returns the product name of the inspected operating system. The product name is generally some freeform string which can be displayed to the user, but should not be parsed by programs.

If the product name could not be determined, then the string `unknown` is returned.

Please read “INSPECTION” in `guestfs` (3) for more details.

### **inspect-get-product-variant**

```
inspect-get-product-variant root
```

This returns the product variant of the inspected operating system.

For Windows guests, this returns the contents of the Registry key `HKLM\Software\Microsoft\Windows NT\CurrentVersion\InstallationType` which is usually a string such as `Client` or `Server` (other values are possible). This can be used to distinguish consumer and enterprise versions of Windows that have the same version number (for example, Windows 7 and Windows 2008 Server are both version 6.1, but the former is `Client` and the latter is `Server`).

For enterprise Linux guests, in future we intend this to return the product variant such as `Desktop`, `Server` and so on. But this is not implemented at present.

If the product variant could not be determined, then the string `unknown` is returned.

Please read “INSPECTION” in `guestfs` (3) for more details. See also “`inspect-get-product-name`”, “`inspect-get-major-version`”.

### **inspect-get-roots**

```
inspect-get-roots
```

This function is a convenient way to get the list of root devices, as returned from a previous call to “`inspect-os`”, but without redoing the whole inspection process.

This returns an empty list if either no root devices were found or the caller has not called “`inspect-os`”.

Please read “INSPECTION” in `guestfs` (3) for more details.

### **inspect-get-type**

```
inspect-get-type root
```

This returns the type of the inspected operating system. Currently defined types are:

“`linux`”

Any Linux-based operating system.

“`windows`”

Any Microsoft Windows operating system.

“`freebsd`”

FreeBSD.

“`netbsd`”

NetBSD.

“`openbsd`”

OpenBSD.

“`hurd`”

GNU/Hurd.

“dos”

MS-DOS, FreeDOS and others.

“minix”

MINIX.

“unknown”

The operating system type could not be determined.

Future versions of libguestfs may return other strings here. The caller should be prepared to handle any string.

Please read “INSPECTION” in **guestfs**(3) for more details.

#### **inspect-get-windows-current-control-set**

```
inspect-get-windows-current-control-set root
```

This returns the Windows CurrentControlSet of the inspected guest. The CurrentControlSet is a registry key name such as ControlSet001.

This call assumes that the guest is Windows and that the Registry could be examined by inspection. If this is not the case then an error is returned.

Please read “INSPECTION” in **guestfs**(3) for more details.

#### **inspect-get-windows-software-hive**

```
inspect-get-windows-software-hive root
```

This returns the path to the hive (binary Windows Registry file) corresponding to HKLM\SOFTWARE.

This call assumes that the guest is Windows and that the guest has a software hive file with the right name. If this is not the case then an error is returned. This call does not check that the hive is a valid Windows Registry hive.

You can use “hivex-open” to read or write to the hive.

Please read “INSPECTION” in **guestfs**(3) for more details.

#### **inspect-get-windows-system-hive**

```
inspect-get-windows-system-hive root
```

This returns the path to the hive (binary Windows Registry file) corresponding to HKLM\SYSTEM.

This call assumes that the guest is Windows and that the guest has a system hive file with the right name. If this is not the case then an error is returned. This call does not check that the hive is a valid Windows Registry hive.

You can use “hivex-open” to read or write to the hive.

Please read “INSPECTION” in **guestfs**(3) for more details.

#### **inspect-get-windows-systemroot**

```
inspect-get-windows-systemroot root
```

This returns the Windows systemroot of the inspected guest. The systemroot is a directory path such as */WINDOWS*.

This call assumes that the guest is Windows and that the systemroot could be determined by inspection. If this is not the case then an error is returned.

Please read “INSPECTION” in **guestfs**(3) for more details.

#### **inspect-is-live**

```
inspect-is-live root
```

This is deprecated and always returns false.

Please read “INSPECTION” in **guestfs**(3) for more details.

*This function is deprecated.* There is no replacement. Consult the API documentation in **guestfs**(3) for

further information.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### **inspect-is-multipart**

```
inspect-is-multipart root
```

This is deprecated and always returns false.

Please read “INSPECTION” in **guestfs**(3) for more details.

*This function is deprecated.* There is no replacement. Consult the API documentation in **guestfs**(3) for further information.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### **inspect-is-netinst**

```
inspect-is-netinst root
```

This is deprecated and always returns false.

Please read “INSPECTION” in **guestfs**(3) for more details.

*This function is deprecated.* There is no replacement. Consult the API documentation in **guestfs**(3) for further information.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### **inspect-list-applications**

```
inspect-list-applications root
```

Return the list of applications installed in the operating system.

*Note:* This call works differently from other parts of the inspection API. You have to call “inspect-os”, then “inspect-get-mountpoints”, then mount up the disks, before calling this. Listing applications is a significantly more difficult operation which requires access to the full filesystem. Also note that unlike the other “inspect-get-\*” calls which are just returning data cached in the libguestfs handle, this call actually reads parts of the mounted filesystems during the call.

This returns an empty list if the inspection code was not able to determine the list of applications.

The application structure contains the following fields:

**app\_name**

The name of the application. For Linux guests, this is the package name.

**app\_display\_name**

The display name of the application, sometimes localized to the install language of the guest operating system.

If unavailable this is returned as an empty string “”. Callers needing to display something can use **app\_name** instead.

**app\_epoch**

For package managers which use epochs, this contains the epoch of the package (an integer). If unavailable, this is returned as 0.

**app\_version**

The version string of the application or package. If unavailable this is returned as an empty string “”.

**app\_release**

The release string of the application or package, for package managers that use this. If unavailable this is returned as an empty string “”.

**app\_install\_path**

The installation path of the application (on operating systems such as Windows which use installation paths). This path is in the format used by the guest operating system, it is not a libguestfs path.

If unavailable this is returned as an empty string "".

**app\_trans\_path**

The install path translated into a libguestfs path. If unavailable this is returned as an empty string "".

**app\_publisher**

The name of the publisher of the application, for package managers that use this. If unavailable this is returned as an empty string "".

**app\_url**

The URL (eg. upstream URL) of the application. If unavailable this is returned as an empty string "".

**app\_source\_package**

For packaging systems which support this, the name of the source package. If unavailable this is returned as an empty string "".

**app\_summary**

A short (usually one line) description of the application or package. If unavailable this is returned as an empty string "".

**app\_description**

A longer description of the application or package. If unavailable this is returned as an empty string "".

Please read “INSPECTION” in **guestfs**(3) for more details.

*This function is deprecated.* In new code, use the “inspect-list-applications2” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

**inspect-list-applications2**

```
inspect-list-applications2 root
```

Return the list of applications installed in the operating system.

*Note:* This call works differently from other parts of the inspection API. You have to call “inspect-os”, then “inspect-get-mountpoints”, then mount up the disks, before calling this. Listing applications is a significantly more difficult operation which requires access to the full filesystem. Also note that unlike the other “inspect-get-\*” calls which are just returning data cached in the libguestfs handle, this call actually reads parts of the mounted filesystems during the call.

This returns an empty list if the inspection code was not able to determine the list of applications.

The application structure contains the following fields:

**app2\_name**

The name of the application. For Linux guests, this is the package name.

**app2\_display\_name**

The display name of the application, sometimes localized to the install language of the guest operating system.

If unavailable this is returned as an empty string "". Callers needing to display something can use **app2\_name** instead.

**app2\_epoch**

For package managers which use epochs, this contains the epoch of the package (an integer). If unavailable, this is returned as 0.



**app2\_version**

The version string of the application or package. If unavailable this is returned as an empty string "".

**app2\_release**

The release string of the application or package, for package managers that use this. If unavailable this is returned as an empty string "".

**app2\_arch**

The architecture string of the application or package, for package managers that use this. If unavailable this is returned as an empty string "".

**app2\_install\_path**

The installation path of the application (on operating systems such as Windows which use installation paths). This path is in the format used by the guest operating system, it is not a libguestfs path.

If unavailable this is returned as an empty string "".

**app2\_trans\_path**

The install path translated into a libguestfs path. If unavailable this is returned as an empty string "".

**app2\_publisher**

The name of the publisher of the application, for package managers that use this. If unavailable this is returned as an empty string "".

**app2\_url**

The URL (eg. upstream URL) of the application. If unavailable this is returned as an empty string "".

**app2\_source\_package**

For packaging systems which support this, the name of the source package. If unavailable this is returned as an empty string "".

**app2\_summary**

A short (usually one line) description of the application or package. If unavailable this is returned as an empty string "".

**app2\_description**

A longer description of the application or package. If unavailable this is returned as an empty string "".

Please read “INSPECTION” in **guestfs**(3) for more details.

**inspect-os****inspect-os**

This function uses other libguestfs functions and certain heuristics to inspect the disk(s) (usually disks belonging to a virtual machine), looking for operating systems.

The list returned is empty if no operating systems were found.

If one operating system was found, then this returns a list with a single element, which is the name of the root filesystem of this operating system. It is also possible for this function to return a list containing more than one element, indicating a dual-boot or multi-boot virtual machine, with each element being the root filesystem of one of the operating systems.

You can pass the root string(s) returned to other “inspect-get-\*” functions in order to query further information about each operating system, such as the name and version.

This function uses other libguestfs features such as “mount-ro” and “umount-all” in order to mount and unmount filesystems and look at the contents. This should be called with no disks currently mounted. The function may also use Augeas, so any existing Augeas handle will be closed.

This function cannot decrypt encrypted disks. The caller must do that first (supplying the necessary keys) if the disk is encrypted.

Please read “INSPECTION” in **guestfs**(3) for more details.

See also “list-fileSYSTEMS”.

### **is-blockdev**

#### **is-blockdev-opts**

```
is-blockdev path [followsymlinks:true|false]
```

This returns `true` if and only if there is a block device with the given path name.

If the optional flag `followsymlinks` is `true`, then a symlink (or chain of symlinks) that ends with a block device also causes the function to return `true`.

This call only looks at files within the guest filesystem. Libguestfs partitions and block devices (eg. `/dev/sda`) cannot be used as the path parameter of this call.

See also “stat”.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

### **is-chardev**

#### **is-chardev-opts**

```
is-chardev path [followsymlinks:true|false]
```

This returns `true` if and only if there is a character device with the given path name.

If the optional flag `followsymlinks` is `true`, then a symlink (or chain of symlinks) that ends with a chardev also causes the function to return `true`.

See also “stat”.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

### **is-config**

```
is-config
```

This returns `true` iff this handle is being configured (in the `CONFIG` state).

For more information on states, see **guestfs**(3).

### **is-dir**

#### **is-dir-opts**

```
is-dir path [followsymlinks:true|false]
```

This returns `true` if and only if there is a directory with the given path name. Note that it returns `false` for other objects like files.

If the optional flag `followsymlinks` is `true`, then a symlink (or chain of symlinks) that ends with a directory also causes the function to return `true`.

See also “stat”.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

### **is-fifo**

#### **is-fifo-opts**

```
is-fifo path [followsymlinks:true|false]
```

This returns `true` if and only if there is a FIFO (named pipe) with the given path name.

If the optional flag `followsymlinks` is `true`, then a symlink (or chain of symlinks) that ends with a FIFO also causes the function to return `true`.

See also “stat”.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

### **is-file**

#### **is-file-opts**

```
is-file path [followsymlinks:true|false]
```

This returns `true` if and only if there is a regular file with the given path name. Note that it returns `false`

for other objects like directories.

If the optional flag `followsymlinks` is true, then a symlink (or chain of symlinks) that ends with a file also causes the function to return true.

See also “stat”.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

### **is-lv**

```
is-lv mountable
```

This command tests whether `mountable` is a logical volume, and returns true iff this is the case.

### **is-socket**

#### **is-socket-opts**

```
is-socket path [followsymlinks:true|false]
```

This returns true if and only if there is a Unix domain socket with the given `path` name.

If the optional flag `followsymlinks` is true, then a symlink (or chain of symlinks) that ends with a socket also causes the function to return true.

See also “stat”.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

### **is-symlink**

```
is-symlink path
```

This returns true if and only if there is a symbolic link with the given `path` name.

See also “stat”.

### **is-whole-device**

```
is-whole-device device
```

This returns true if and only if `device` refers to a whole block device. That is, not a partition or a logical device.

### **is-zero**

```
is-zero path
```

This returns true iff the file exists and the file is empty or it contains all zero bytes.

### **is-zero-device**

```
is-zero-device device
```

This returns true iff the device exists and contains all zero bytes.

Note that for large devices this can take a long time to run.

### **isoinfo**

```
isoinfo isofile
```

This is the same as “isoinfo-device” except that it works for an ISO file located inside some other mounted filesystem. Note that in the common case where you have added an ISO file as a libguestfs device, you would *not* call this. Instead you would call “isoinfo-device”.

### **isoinfo-device**

```
isoinfo-device device
```

`device` is an ISO device. This returns a struct of information read from the primary volume descriptor (the ISO equivalent of the superblock) of the device.

Usually it is more efficient to use the **isoinfo**(1) command with the `-d` option on the host to analyze ISO files, instead of going through libguestfs.

For information on the primary volume descriptor fields, see [https://wiki.osdev.org/ISO\\_9660#The\\_Primary\\_Volume\\_Descriptor](https://wiki.osdev.org/ISO_9660#The_Primary_Volume_Descriptor)

**journal-close**

```
journal-close
```

Close the journal handle.

This command depends on the feature `journal`. See also “feature-available”.

**journal-get**

```
journal-get
```

Read the current journal entry. This returns all the fields in the journal as a set of `(attrname, attrval)` pairs. The `attrname` is the field name (a string).

The `attrval` is the field value (a binary blob, often but not always a string). Please note that `attrval` is a byte array, *not* a `\0`-terminated C string.

The length of data may be truncated to the data threshold (see: “journal-set-data-threshold”, “journal-get-data-threshold”).

If you set the data threshold to unlimited (0) then this call can read a journal entry of any size, ie. it is not limited by the libguestfs protocol.

This command depends on the feature `journal`. See also “feature-available”.

**journal-get-data-threshold**

```
journal-get-data-threshold
```

Get the current data threshold for reading journal entries. This is a hint to the journal that it may truncate data fields to this size when reading them (note also that it may not truncate them). If this returns 0, then the threshold is unlimited.

See also “journal-set-data-threshold”.

This command depends on the feature `journal`. See also “feature-available”.

**journal-get-realtime-usec**

```
journal-get-realtime-usec
```

Get the realtime (wallclock) timestamp of the current journal entry.

This command depends on the feature `journal`. See also “feature-available”.

**journal-next**

```
journal-next
```

Move to the next journal entry. You have to call this at least once after opening the handle before you are able to read data.

The returned boolean tells you if there are any more journal records to read. `true` means you can read the next record (eg. using “journal-get”), and `false` means you have reached the end of the journal.

This command depends on the feature `journal`. See also “feature-available”.

**journal-open**

```
journal-open directory
```

Open the systemd journal located in *directory*. Any previously opened journal handle is closed.

The contents of the journal can be read using “journal-next” and “journal-get”.

After you have finished using the journal, you should close the handle by calling “journal-close”.

This command depends on the feature `journal`. See also “feature-available”.

**journal-set-data-threshold**

```
journal-set-data-threshold threshold
```

Set the data threshold for reading journal entries. This is a hint to the journal that it may truncate data fields to this size when reading them (note also that it may not truncate them). If you set this to 0, then the threshold is unlimited.

See also “journal-get-data-threshold”.

This command depends on the feature `journal`. See also “feature-available”.

### **journal-skip**

`journal-skip skip`

Skip forwards (`skip ≥ 0`) or backwards (`skip < 0`) in the journal.

The number of entries actually skipped is returned (note `rskip ≥ 0`). If this is not the same as the absolute value of the skip parameter (`|skip|`) you passed in then it means you have reached the end or the start of the journal.

This command depends on the feature `journal`. See also “feature-available”.

### **kill-subprocess**

`kill-subprocess`

This kills the hypervisor.

Do not call this. See: “shutdown” instead.

*This function is deprecated.* In new code, use the “shutdown” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### **launch**

#### **run**

`launch`

You should call this after configuring the handle (eg. adding drives) but before performing any actions.

Do not call “launch” twice on the same handle. Although it will not give an error (for historical reasons), the precise behaviour when you do this is not well defined. Handles are very cheap to create, so create a new one for each launch.

### **lchown**

`lchown owner group path`

Change the file owner to `owner` and group to `group`. This is like “chown” but if `path` is a symlink then the link itself is changed, not the target.

Only numeric uid and gid are supported. If you want to use names, you will need to locate and parse the password file yourself (Augeas support makes this relatively easy).

### **ldmtool-create-all**

`ldmtool-create-all`

This function scans all block devices looking for Windows dynamic disk volumes and partitions, and creates devices for any that were found.

Call “list-ldm-volumes” and “list-ldm-partitions” to return all devices.

Note that you **don’t** normally need to call this explicitly, since it is done automatically at “launch” time. However you might want to call this function if you have hotplugged disks or have just created a Windows dynamic disk.

This command depends on the feature `ldm`. See also “feature-available”.

### **ldmtool-diskgroup-disks**

`ldmtool-diskgroup-disks diskgroup`

Return the disks in a Windows dynamic disk group. The `diskgroup` parameter should be the GUID of a disk group, one element from the list returned by “ldmtool-scan”.

This command depends on the feature `ldm`. See also “feature-available”.

**ldmtool-diskgroup-name**

```
ldmtool-diskgroup-name diskgroup
```

Return the name of a Windows dynamic disk group. The `diskgroup` parameter should be the GUID of a disk group, one element from the list returned by “`ldmtool-scan`”.

This command depends on the feature `ldm`. See also “feature-available”.

**ldmtool-diskgroup-volumes**

```
ldmtool-diskgroup-volumes diskgroup
```

Return the volumes in a Windows dynamic disk group. The `diskgroup` parameter should be the GUID of a disk group, one element from the list returned by “`ldmtool-scan`”.

This command depends on the feature `ldm`. See also “feature-available”.

**ldmtool-remove-all**

```
ldmtool-remove-all
```

This is essentially the opposite of “`ldmtool-create-all`”. It removes the device mapper mappings for all Windows dynamic disk volumes.

This command depends on the feature `ldm`. See also “feature-available”.

**ldmtool-scan**

```
ldmtool-scan
```

This function scans for Windows dynamic disks. It returns a list of identifiers (GUIDs) for all disk groups that were found. These identifiers can be passed to other “`ldmtool-*`” functions.

This function scans all block devices. To scan a subset of block devices, call “`ldmtool-scan-devices`” instead.

This command depends on the feature `ldm`. See also “feature-available”.

**ldmtool-scan-devices**

```
ldmtool-scan-devices 'devices ...'
```

This function scans for Windows dynamic disks. It returns a list of identifiers (GUIDs) for all disk groups that were found. These identifiers can be passed to other “`ldmtool-*`” functions.

The parameter `devices` is a list of block devices which are scanned. If this list is empty, all block devices are scanned.

This command depends on the feature `ldm`. See also “feature-available”.

**ldmtool-volume-hint**

```
ldmtool-volume-hint diskgroup volume
```

Return the hint field of the volume named `volume` in the disk group with GUID `diskgroup`. This may not be defined, in which case the empty string is returned. The hint field is often, though not always, the name of a Windows drive, eg. `E:`.

This command depends on the feature `ldm`. See also “feature-available”.

**ldmtool-volume-partitions**

```
ldmtool-volume-partitions diskgroup volume
```

Return the list of partitions in the volume named `volume` in the disk group with GUID `diskgroup`.

This command depends on the feature `ldm`. See also “feature-available”.

**ldmtool-volume-type**

```
ldmtool-volume-type diskgroup volume
```

Return the type of the volume named `volume` in the disk group with GUID `diskgroup`.

Possible volume types that can be returned here include: `simple`, `spanned`, `striped`, `mirrored`, `raid5`. Other types may also be returned.

This command depends on the feature `ldm`. See also “feature-available”.

### **lgetxattr**

```
lgetxattr path name
```

Get a single extended attribute from file `path` named `name`. If `path` is a symlink, then this call returns an extended attribute from the symlink.

Normally it is better to get all extended attributes from a file in one go by calling “`getxattrs`”. However some Linux filesystem implementations are buggy and do not provide a way to list out attributes. For these filesystems (notably `ntfs-3g`) you have to know the names of the extended attributes you want in advance and call this function.

Extended attribute values are blobs of binary data. If there is no extended attribute named `name`, this returns an error.

See also: “`lgetxattrs`”, “`getxattr`”, `attr(5)`.

This command depends on the feature `linuxxattrs`. See also “feature-available”.

### **lgetxattrs**

```
lgetxattrs path
```

This is the same as “`getxattrs`”, but if `path` is a symbolic link, then it returns the extended attributes of the link itself.

This command depends on the feature `linuxxattrs`. See also “feature-available”.

### **list-9p**

```
list-9p
```

List all 9p filesystems attached to the guest. A list of mount tags is returned.

### **list-devices**

```
list-devices
```

List all the block devices.

The full block device names are returned, eg. `/dev/sda`.

See also “`list-filesystems`”.

### **list-disk-labels**

```
list-disk-labels
```

If you add drives using the optional `label` parameter of “`add-drive-opts`”, you can use this call to map between disk labels, and raw block device and partition names (like `/dev/sda` and `/dev/sda1`).

This returns a hashtable, where keys are the disk labels (*without* the `/dev/disk/guestfs` prefix), and the values are the full raw block device and partition names (eg. `/dev/sda` and `/dev/sda1`).

### **list-dm-devices**

```
list-dm-devices
```

List all device mapper devices.

The returned list contains `/dev/mapper/*` devices, eg. ones created by a previous call to “`luks-open`”.

Device mapper devices which correspond to logical volumes are *not* returned in this list. Call “`lvs`” if you want to list logical volumes.

### **list-filesystems**

```
list-filesystems
```

This inspection command looks for filesystems on partitions, block devices and logical volumes, returning a list of mountables containing filesystems and their type.

The return value is a hash, where the keys are the devices containing filesystems, and the values are the filesystem types. For example:

```
"/dev/sda1" => "ntfs"
"/dev/sda2" => "ext2"
"/dev/vg_guest/lv_root" => "ext4"
"/dev/vg_guest/lv_swap" => "swap"
```

The key is not necessarily a block device. It may also be an opaque ‘mountable’ string which can be passed to “mount”.

The value can have the special value “unknown”, meaning the content of the device is undetermined or empty. “swap” means a Linux swap partition.

In libguestfs  $\leq 1.36$  this command ran other libguestfs commands, which might have included “mount” and “umount”, and therefore you had to use this soon after launch and only when nothing else was mounted. This restriction is removed in libguestfs  $\geq 1.38$ .

Not all of the filesystems returned will be mountable. In particular, swap partitions are returned in the list. Also this command does not check that each filesystem found is valid and mountable, and some filesystems might be mountable but require special options. Filesystems may not all belong to a single logical operating system (use “inspect-os” to look for OSes).

### list-ldm-partitions

```
list-ldm-partitions
```

This function returns all Windows dynamic disk partitions that were found at launch time. It returns a list of device names.

This command depends on the feature ldm. See also “feature-available”.

### list-ldm-volumes

```
list-ldm-volumes
```

This function returns all Windows dynamic disk volumes that were found at launch time. It returns a list of device names.

This command depends on the feature ldm. See also “feature-available”.

### list-md-devices

```
list-md-devices
```

List all Linux md devices.

### list-partitions

```
list-partitions
```

List all the partitions detected on all block devices.

The full partition device names are returned, eg. */dev/sda1*

This does not return logical volumes. For that you will need to call “lvs”.

See also “list-filesystems”.

## ll

```
ll directory
```

List the files in *directory* (relative to the root directory, there is no cwd) in the format of `ls -la`.

This command is mostly useful for interactive sessions. It is *not* intended that you try to parse the output string.

## llz

```
llz directory
```

List the files in *directory* in the format of `ls -laZ`.

This command is mostly useful for interactive sessions. It is *not* intended that you try to parse the output string.

*This function is deprecated.* In new code, use the “lgetxattrs” call instead.



Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### **ln**

```
ln target linkname
```

This command creates a hard link.

### **ln-f**

```
ln-f target linkname
```

This command creates a hard link, removing the link `linkname` if it exists already.

### **ln-s**

```
ln-s target linkname
```

This command creates a symbolic link using the `ln -s` command.

### **ln-sf**

```
ln-sf target linkname
```

This command creates a symbolic link using the `ln -sf` command, The `-f` option removes the link (`linkname`) if it exists already.

### **lremovexattr**

```
lremovexattr xattr path
```

This is the same as “`removexattr`”, but if `path` is a symbolic link, then it removes an extended attribute of the link itself.

This command depends on the feature `linuxxattrs`. See also “feature-available”.

### **ls**

```
ls directory
```

List the files in *directory* (relative to the root directory, there is no `cwd`). The `.` and `..` entries are not returned, but hidden files are shown.

### **ls0**

```
ls0 dir (filenames|-)
```

This specialized command is used to get a listing of the filenames in the directory `dir`. The list of filenames is written to the local file *filenames* (on the host).

In the output file, the filenames are separated by `\0` characters.

`.` and `..` are not returned. The filenames are not sorted.

Use `-` instead of a filename to read/write from `stdin/stdout`.

### **lsetxattr**

```
lsetxattr xattr val vallen path
```

This is the same as “`setxattr`”, but if `path` is a symbolic link, then it sets an extended attribute of the link itself.

This command depends on the feature `linuxxattrs`. See also “feature-available”.

### **lstat**

```
lstat path
```

Returns file information for the given `path`.

This is the same as “`stat`” except that if `path` is a symbolic link, then the link is stat-ed, not the file it refers to.

This is the same as the **lstat**(2) system call.

*This function is deprecated.* In new code, use the “`lstatns`” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that

there are problems with correct use of these functions.

### **lstatlist**

```
lstatlist path 'names ...'
```

This call allows you to perform the “lstat” operation on multiple files, where all files are in the directory `path`. `names` is the list of files from this directory.

On return you get a list of stat structs, with a one-to-one correspondence to the `names` list. If any name did not exist or could not be lstat'd, then the `st_ino` field of that structure is set to `-1`.

This call is intended for programs that want to efficiently list a directory contents without making many round-trips. See also “lxattrlist” for a similarly efficient call for getting extended attributes.

*This function is deprecated.* In new code, use the “lstatnslist” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### **lstatns**

```
lstatns path
```

Returns file information for the given path.

This is the same as “statns” except that if `path` is a symbolic link, then the link is stat-ed, not the file it refers to.

This is the same as the **lstat** (2) system call.

### **lstatnslist**

```
lstatnslist path 'names ...'
```

This call allows you to perform the “lstatns” operation on multiple files, where all files are in the directory `path`. `names` is the list of files from this directory.

On return you get a list of stat structs, with a one-to-one correspondence to the `names` list. If any name did not exist or could not be lstat'd, then the `st_ino` field of that structure is set to `-1`.

This call is intended for programs that want to efficiently list a directory contents without making many round-trips. See also “lxattrlist” for a similarly efficient call for getting extended attributes.

### **luks-add-key**

```
luks-add-key device keyslot
```

This command adds a new key on LUKS device `device`. `key` is any existing key, and is used to access the device. `newkey` is the new key to add. `keyslot` is the key slot that will be replaced.

Note that if `keyslot` already contains a key, then this command will fail. You have to use “luks-kill-slot” first to remove that key.

This command has one or more key or passphrase parameters. Guestfish will prompt for these separately.

This command depends on the feature `luks`. See also “feature-available”.

### **luks-close**

```
luks-close device
```

This closes a LUKS device that was created earlier by “luks-open” or “luks-open-ro”. The `device` parameter must be the name of the LUKS mapping device (ie. `/dev/mapper/mapname`) and *not* the name of the underlying block device.

*This function is deprecated.* In new code, use the “cryptsetup-close” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

This command depends on the feature `luks`. See also “feature-available”.

**luks-format**

```
luks-format device keyslot
```

This command erases existing data on `device` and formats the device as a LUKS encrypted device. `key` is the initial key, which is added to key slot `keyslot`. (LUKS supports 8 key slots, numbered 0–7).

This command has one or more `key` or `passphrase` parameters. Guestfish will prompt for these separately.

This command depends on the feature `luks`. See also “feature-available”.

**luks-format-cipher**

```
luks-format-cipher device keyslot cipher
```

This command is the same as “`luks-format`” but it also allows you to set the `cipher` used.

This command has one or more `key` or `passphrase` parameters. Guestfish will prompt for these separately.

This command depends on the feature `luks`. See also “feature-available”.

**luks-kill-slot**

```
luks-kill-slot device keyslot
```

This command deletes the key in key slot `keyslot` from the encrypted LUKS device `device`. `key` must be one of the *other* keys.

This command has one or more `key` or `passphrase` parameters. Guestfish will prompt for these separately.

This command depends on the feature `luks`. See also “feature-available”.

**luks-open**

```
luks-open device mapname
```

This command opens a block device which has been encrypted according to the Linux Unified Key Setup (LUKS) standard.

`device` is the encrypted block device or partition.

The caller must supply one of the keys associated with the LUKS block device, in the `key` parameter.

This creates a new block device called `/dev/mapper/mapname`. Reads and writes to this block device are decrypted from and encrypted to the underlying `device` respectively.

If this block device contains LVM volume groups, then calling “`lvm-scan`” with the `activate` parameter `true` will make them visible.

Use “`list-dm-devices`” to list all device mapper devices.

This command has one or more `key` or `passphrase` parameters. Guestfish will prompt for these separately.

*This function is deprecated.* In new code, use the “`cryptsetup-open`” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

This command depends on the feature `luks`. See also “feature-available”.

**luks-open-ro**

```
luks-open-ro device mapname
```

This is the same as “`luks-open`” except that a read-only mapping is created.

This command has one or more `key` or `passphrase` parameters. Guestfish will prompt for these separately.

*This function is deprecated.* In new code, use the “`cryptsetup-open`” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

This command depends on the feature `luks`. See also “feature-available”.

**luks-uuid**

```
luks-uuid device
```

This returns the UUID of the LUKS device device.

This command depends on the feature luks. See also “feature-available”.

**lvcreate**

```
lvcreate logvol volgroup mbytes
```

This creates an LVM logical volume called logvol on the volume group volgroup, with size megabytes.

This command depends on the feature lvm2. See also “feature-available”.

**lvcreate-free**

```
lvcreate-free logvol volgroup percent
```

Create an LVM logical volume called */dev/volgroup/logvol*, using approximately percent % of the free space remaining in the volume group. Most usefully, when percent is 100 this will create the largest possible LV.

This command depends on the feature lvm2. See also “feature-available”.

**lvm-canonical-lv-name**

```
lvm-canonical-lv-name lvname
```

This converts alternative naming schemes for LVs that you might find to the canonical name. For example, */dev/mapper/VG-LV* is converted to */dev/VG/LV*.

This command returns an error if the lvname parameter does not refer to a logical volume. In this case errno will be set to EINVAL.

See also “is-lv”, “canonical-device-name”.

**lvm-clear-filter**

```
lvm-clear-filter
```

This undoes the effect of “lvm-set-filter”. LVM will be able to see every block device.

This command also clears the LVM cache and performs a volume group scan.

**lvm-remove-all**

```
lvm-remove-all
```

This command removes all LVM logical volumes, volume groups and physical volumes.

This command depends on the feature lvm2. See also “feature-available”.

**lvm-scan**

```
lvm-scan true|false
```

This scans all block devices and rebuilds the list of LVM physical volumes, volume groups and logical volumes.

If the activate parameter is true then newly found volume groups and logical volumes are activated, meaning the LV */dev/VG/LV* devices become visible.

When a libguestfs handle is launched it scans for existing devices, so you do not normally need to use this API. However it is useful when you have added a new device or deleted an existing device (such as when the “luks-open” API is used).

**lvm-set-filter**

```
lvm-set-filter 'devices ...'
```

This sets the LVM device filter so that LVM will only be able to “see” the block devices in the list devices, and will ignore all other attached block devices.

Where disk image(s) contain duplicate PVs or VGs, this command is useful to get LVM to ignore the

duplicates, otherwise LVM can get confused. Note also there are two types of duplication possible: either cloned PVs/VGs which have identical UUIDs; or VGs that are not cloned but just happen to have the same name. In normal operation you cannot create this situation, but you can do it outside LVM, eg. by cloning disk images or by bit twiddling inside the LVM metadata.

This command also clears the LVM cache and performs a volume group scan.

You can filter whole block devices or individual partitions.

You cannot use this if any VG is currently in use (eg. contains a mounted filesystem), even if you are not filtering out that VG.

This command depends on the feature `lvm2`. See also “feature-available”.

### **lvremove**

```
lvremove device
```

Remove an LVM logical volume `device`, where `device` is the path to the LV, such as `/dev/VG/LV`.

You can also remove all LVs in a volume group by specifying the VG name, `/dev/VG`.

This command depends on the feature `lvm2`. See also “feature-available”.

### **lvrename**

```
lvrename logvol newlogvol
```

Rename a logical volume `logvol` with the new name `newlogvol`.

### **lvresize**

```
lvresize device mbytes
```

This resizes (expands or shrinks) an existing LVM logical volume to `mbytes`. When reducing, data in the reduced part is lost.

This command depends on the feature `lvm2`. See also “feature-available”.

### **lvresize-free**

```
lvresize-free lv percent
```

This expands an existing logical volume `lv` so that it fills `pc` % of the remaining free space in the volume group. Commonly you would call this with `pc = 100` which expands the logical volume as much as possible, using all remaining free space in the volume group.

This command depends on the feature `lvm2`. See also “feature-available”.

### **lvs**

```
lvs
```

List all the logical volumes detected. This is the equivalent of the **lvs**(8) command.

This returns a list of the logical volume device names (eg. `/dev/VolGroup00/LogVol00`).

See also “lvs-full”, “list-filestystems”.

This command depends on the feature `lvm2`. See also “feature-available”.

### **lvs-full**

```
lvs-full
```

List all the logical volumes detected. This is the equivalent of the **lvs**(8) command. The “full” version includes all fields.

This command depends on the feature `lvm2`. See also “feature-available”.

### **lvuuid**

```
lvuuid device
```

This command returns the UUID of the LVM LV `device`.

**lxattrlist**

```
lxattrlist path 'names ...'
```

This call allows you to get the extended attributes of multiple files, where all files are in the directory path. names is the list of files from this directory.

On return you get a flat list of xattr structs which must be interpreted sequentially. The first xattr struct always has a zero-length attrname. attrval in this struct is zero-length to indicate there was an error doing “lgetxattr” for this file, *or* is a C string which is a decimal number (the number of following attributes for this file, which could be "0"). Then after the first xattr struct are the zero or more attributes for the first named file. This repeats for the second and subsequent files.

This call is intended for programs that want to efficiently list a directory contents without making many round-trips. See also “lstatlist” for a similarly efficient call for getting standard stats.

This command depends on the feature linuxxattrs. See also “feature-available”.

**max-disks**

```
max-disks
```

Return the maximum number of disks that may be added to a handle (eg. by “add-drive-opts” and similar calls).

This function was added in libguestfs 1.19.7. In previous versions of libguestfs the limit was 25.

See “MAXIMUM NUMBER OF DISKS” in **guestfs** (3) for additional information on this topic.

**md-create**

```
md-create name 'devices ...' [missingbitmap:N] [nrdevices:N] [spare:N] [chunk:N]
```

Create a Linux md (RAID) device named name on the devices in the list devices.

The optional parameters are:

missingbitmap

A bitmap of missing devices. If a bit is set it means that a missing device is added to the array. The least significant bit corresponds to the first device in the array.

As examples:

If devices = ["/dev/sda"] and missingbitmap = 0x1 then the resulting array would be [<missing>, "/dev/sda"].

If devices = ["/dev/sda"] and missingbitmap = 0x2 then the resulting array would be ["/dev/sda", <missing>].

This defaults to 0 (no missing devices).

The length of devices + the number of bits set in missingbitmap must equal nrdevices + spare.

nrdevices

The number of active RAID devices.

If not set, this defaults to the length of devices plus the number of bits set in missingbitmap.

spare

The number of spare devices.

If not set, this defaults to 0.

chunk

The chunk size in bytes.

level

The RAID level, which can be one of: linear, raid0, 0, stripe, raid1, 1, mirror, raid4, 4, raid5, 5, raid6, 6, raid10, 10. Some of these are synonymous, and more levels may be added in future.

If not set, this defaults to `raid1`.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

This command depends on the feature `mdadm`. See also “feature-available”.

### **md-detail**

`md-detail md`

This command exposes the output of `mdadm -DY <md>`. The following fields are usually present in the returned hash. Other fields may also be present.

`level`

The raid level of the MD device.

`devices`

The number of underlying devices in the MD device.

`metadata`

The metadata version used.

`uuid`

The UUID of the MD device.

`name`

The name of the MD device.

This command depends on the feature `mdadm`. See also “feature-available”.

### **md-stat**

`md-stat md`

This call returns a list of the underlying devices which make up the single software RAID array device `md`.

To get a list of software RAID devices, call “list-md-devices”.

Each structure returned corresponds to one device along with additional status information:

`mdstat_device`

The name of the underlying device.

`mdstat_index`

The index of this device within the array.

`mdstat_flags`

Flags associated with this device. This is a string containing (in no specific order) zero or more of the following flags:

W write-mostly

F device is faulty

S device is a RAID spare

R replacement

This command depends on the feature `mdadm`. See also “feature-available”.

### **md-stop**

`md-stop md`

This command deactivates the MD array named `md`. The device is stopped, but it is not destroyed or zeroed.

This command depends on the feature `mdadm`. See also “feature-available”.

### **mkdir**

`mkdir path`

Create a directory named `path`.

**mkdir-mode**

```
mkdir-mode path mode
```

This command creates a directory, setting the initial permissions of the directory to mode.

For common Linux filesystems, the actual mode which is set will be `mode & ~umask & 01777`. Non-native-Linux filesystems may interpret the mode in other ways.

See also “mkdir”, “umask”

**mkdir-p**

```
mkdir-p path
```

Create a directory named path, creating any parent directories as necessary. This is like the `mkdir -p` shell command.

**mkdtemp**

```
mkdtemp tmpl
```

This command creates a temporary directory. The `tmpl` parameter should be a full pathname for the temporary directory name with the final six characters being “XXXXXX”.

For example: “/tmp/myprogXXXXXX” or “/Temp/myprogXXXXXX”, the second one being suitable for Windows filesystems.

The name of the temporary directory that was created is returned.

The temporary directory is created with mode 0700 and is owned by root.

The caller is responsible for deleting the temporary directory and its contents after use.

See also: **mkdtemp**(3)

**mke2fs**

```
mke2fs device [blockcount:N] [blocksize:N] [fragsize:N] [blocksgroup:N] [num]
```

`mke2fs` is used to create an ext2, ext3, or ext4 filesystem on device.

The optional `blockcount` is the size of the filesystem in blocks. If omitted it defaults to the size of device. Note if the filesystem is too small to contain a journal, `mke2fs` will silently create an ext2 filesystem instead.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

**mke2fs-J**

```
mke2fs-J fstype blocksize device journal
```

This creates an ext2/3/4 filesystem on device with an external journal on journal. It is equivalent to the command:

```
mke2fs -t fstype -b blocksize -J device=<journal> <device>
```

See also “mke2journal”.

*This function is deprecated.* In new code, use the “mke2fs” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

**mke2fs-JL**

```
mke2fs-JL fstype blocksize device label
```

This creates an ext2/3/4 filesystem on device with an external journal on the journal labeled label.

See also “mke2journal-L”.

*This function is deprecated.* In new code, use the “mke2fs” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.



**mke2fs-JU**

```
mke2fs-JU fstype blocksize device uuid
```

This creates an ext2/3/4 filesystem on device with an external journal on the journal with UUID uuid.

See also “mke2journal-U”.

*This function is deprecated.* In new code, use the “mke2fs” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

This command depends on the feature linuxfsuuid. See also “feature-available”.

**mke2journal**

```
mke2journal blocksize device
```

This creates an ext2 external journal on device. It is equivalent to the command:

```
mke2fs -O journal_dev -b blocksize device
```

*This function is deprecated.* In new code, use the “mke2fs” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

**mke2journal-L**

```
mke2journal-L blocksize label device
```

This creates an ext2 external journal on device with label label.

*This function is deprecated.* In new code, use the “mke2fs” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

**mke2journal-U**

```
mke2journal-U blocksize uuid device
```

This creates an ext2 external journal on device with UUID uuid.

*This function is deprecated.* In new code, use the “mke2fs” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

This command depends on the feature linuxfsuuid. See also “feature-available”.

**mkfifo**

```
mkfifo mode path
```

This call creates a FIFO (named pipe) called path with mode mode. It is just a convenient wrapper around “mknod”.

Unlike with “mknod”, mode **must** contain only permissions bits.

The mode actually set is affected by the umask.

This command depends on the feature mknod. See also “feature-available”.

**mkfs****mkfs-opts**

```
mkfs fstype device [blocksize:N] [features:...] [inode:N] [sectorsize:N] [label:.]
```

This function creates a filesystem on device. The filesystem type is fstype, for example ext3.

The optional arguments are:

**blocksize**

The filesystem block size. Supported block sizes depend on the filesystem type, but typically they are 1024, 2048 or 4096 for Linux ext2/3 filesystems.

For VFAT and NTFS the `blocksize` parameter is treated as the requested cluster size.

For UFS block sizes, please see **mkfs.ufs** (8).

#### features

This passes the `-O` parameter to the external **mkfs** program.

For certain filesystem types, this allows extra filesystem features to be selected. See **mke2fs** (8) and **mkfs.ufs** (8) for more details.

You cannot use this optional parameter with the `gfs` or `gfs2` filesystem type.

#### inode

This passes the `-I` parameter to the external **mke2fs** (8) program which sets the inode size (only for ext2/3/4 filesystems at present).

#### sectorsize

This passes the `-S` parameter to external **mkfs.ufs** (8) program, which sets sector size for ufs filesystem.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

### mkfs-b

```
mkfs-b fstype blocksize device
```

This call is similar to “mkfs”, but it allows you to control the block size of the resulting filesystem. Supported block sizes depend on the filesystem type, but typically they are 1024, 2048 or 4096 only.

For VFAT and NTFS the `blocksize` parameter is treated as the requested cluster size.

*This function is deprecated.* In new code, use the “mkfs” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### mkfs-btrfs

```
mkfs-btrfs 'devices ...' [allocstart:N] [bytecount:N] [datatype:...] [leafsize:N]
```

Create a btrfs filesystem, allowing all configurables to be set. For more information on the optional arguments, see **mkfs.btrfs** (8).

Since btrfs filesystems can span multiple devices, this takes a non-empty list of devices.

To create general filesystems, use “mkfs”.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

This command depends on the feature `btrfs`. See also “feature-available”.

### mklost-and-found

```
mklost-and-found mountpoint
```

Make the `lost+found` directory, normally in the root directory of an ext2/3/4 filesystem. `mountpoint` is the directory under which we try to create the `lost+found` directory.

### mkmountpoint

```
mkmountpoint exemptpath
```

“mkmountpoint” and “rmmountpoint” are specialized calls that can be used to create extra mountpoints before mounting the first filesystem.

These calls are *only* necessary in some very limited circumstances, mainly the case where you want to mount a mix of unrelated and/or read-only filesystems together.

For example, live CDs often contain a “Russian doll” nest of filesystems, an ISO outer layer, with a squashfs image inside, with an ext2/3 image inside that. You can unpack this as follows in guestfish:

```
add-ro Fedora-11-i686-Live.iso
run
mkmountpoint /cd
mkmountpoint /sqsh
mkmountpoint /ext3fs
mount /dev/sda /cd
mount-loop /cd/LiveOS/squashfs.img /sqsh
mount-loop /sqsh/LiveOS/ext3fs.img /ext3fs
```

The inner filesystem is now unpacked under the /ext3fs mountpoint.

“mkmountpoint” is not compatible with “umount-all”. You may get unexpected errors if you try to mix these calls. It is safest to manually unmount filesystems and remove mountpoints after use.

“umount-all” unmounts filesystems by sorting the paths longest first, so for this to work for manual mountpoints, you must ensure that the innermost mountpoints have the longest pathnames, as in the example code above.

For more details see [https://bugzilla.redhat.com/show\\_bug.cgi?id=599503](https://bugzilla.redhat.com/show_bug.cgi?id=599503)

Autosync [see “set-autosync”, this is set by default on handles] can cause “umount-all” to be called when the handle is closed which can also trigger these issues.

### mknod

```
mknod mode devmajor devminor path
```

This call creates block or character special devices, or named pipes (FIFOs).

The mode parameter should be the mode, using the standard constants. devmajor and devminor are the device major and minor numbers, only used when creating block and character special devices.

Note that, just like **mknod**(2), the mode must be bitwise OR’d with S\_IFBLK, S\_IFCHR, S\_IFIFO or S\_IFSOCK (otherwise this call just creates a regular file). These constants are available in the standard Linux header files, or you can use “mknod-b”, “mknod-c” or “mkfifo” which are wrappers around this command which bitwise OR in the appropriate constant for you.

The mode actually set is affected by the umask.

This command depends on the feature mknod. See also “feature-available”.

### mknod-b

```
mknod-b mode devmajor devminor path
```

This call creates a block device node called path with mode mode and device major/minor devmajor and devminor. It is just a convenient wrapper around “mknod”.

Unlike with “mknod”, mode **must** contain only permissions bits.

The mode actually set is affected by the umask.

This command depends on the feature mknod. See also “feature-available”.

### mknod-c

```
mknod-c mode devmajor devminor path
```

This call creates a char device node called path with mode mode and device major/minor devmajor and devminor. It is just a convenient wrapper around “mknod”.

Unlike with “mknod”, mode **must** contain only permissions bits.

The mode actually set is affected by the umask.

This command depends on the feature mknod. See also “feature-available”.

### mk squashfs

```
mk squashfs path (filename|-) [compress:...] [excludes:...]
```

Create a squashfs filesystem for the specified path.

The optional `compress` flag controls compression. If not given, then the output compressed using `gzip`. Otherwise one of the following strings may be given to select the compression type of the squashfs: `gzip`, `lzma`, `lzo`, `lz4`, `xz`.

The other optional arguments are:

`excludes`

A list of wildcards. Files are excluded if they match any of the wildcards.

Please note that this API may fail when used to compress directories with large files, such as the resulting squashfs will be over 3GB big.

Use – instead of a filename to read/write from stdin/stdout.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

This command depends on the feature `squashfs`. See also “feature-available”.

## **mkswap**

### **mkswap-opts**

```
mkswap device [label:..] [uuid:..]
```

Create a Linux swap partition on device.

The option arguments `label` and `uuid` allow you to set the label and/or UUID of the new swap partition.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

### **mkswap-L**

```
mkswap-L label device
```

Create a swap partition on device with label `label`.

Note that you cannot attach a swap label to a block device (eg. `/dev/sda`), just to a partition. This appears to be a limitation of the kernel or swap tools.

*This function is deprecated.* In new code, use the “mkswap” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### **mkswap-U**

```
mkswap-U uuid device
```

Create a swap partition on device with UUID `uuid`.

*This function is deprecated.* In new code, use the “mkswap” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

This command depends on the feature `linuxfsuuid`. See also “feature-available”.

### **mkswap-file**

```
mkswap-file path
```

Create a swap file.

This command just writes a swap file signature to an existing file. To create the file itself, use something like “fallocate”.

## **mktemp**

```
mktemp tmp1 [suffix:..]
```

This command creates a temporary file. The `tmp1` parameter should be a full pathname for the temporary directory name with the final six characters being “XXXXXX”.

For example: “/tmp/myprogXXXXXX” or “/Temp/myprogXXXXXX”, the second one being suitable for Windows filesystems.

The name of the temporary file that was created is returned.

The temporary file is created with mode 0600 and is owned by root.

The caller is responsible for deleting the temporary file after use.

If the optional `suffix` parameter is given, then the suffix (eg. `.txt`) is appended to the temporary name.

See also: “`mkdtemp`”.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

### **modprobe**

```
modprobe modulename
```

This loads a kernel module in the appliance.

This command depends on the feature `linuxmodules`. See also “feature-available”.

### **mount**

```
mount mountable mountpoint
```

Mount a guest disk at a position in the filesystem. Block devices are named `/dev/sda`, `/dev/sdb` and so on, as they were added to the guest. If those block devices contain partitions, they will have the usual names (eg. `/dev/sda1`). Also LVM `/dev/VG/LV`-style names can be used, or ‘mountable’ strings returned by “`list-fileSystems`” or “`inspect-get-mountpoints`”.

The rules are the same as for **mount**(2): A filesystem must first be mounted on/ before others can be mounted. Other filesystems can only be mounted on directories which already exist.

The mounted filesystem is writable, if we have sufficient permissions on the underlying device.

Before `libguestfs` 1.13.16, this call implicitly added the options `sync` and `noatime`. The `sync` option greatly slowed writes and caused many problems for users. If your program might need to work with older versions of `libguestfs`, use “`mount-options`” instead (using an empty string for the first parameter if you don’t want any options).

### **mount-9p**

```
mount-9p mounttag mountpoint [options:..]
```

Mount the `virtio-9p` filesystem with the tag `mounttag` on the directory `mountpoint`.

If required, `trans=virtio` will be automatically added to the options. Any other options required can be passed in the optional `options` parameter.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

### **mount-local**

```
mount-local localmountpoint [readonly:true|false] [options:..] [cachetimeout:N]
```

This call exports the `libguestfs`-accessible filesystem to a local mountpoint (directory) called `localmountpoint`. Ordinary reads and writes to files and directories under `localmountpoint` are redirected through `libguestfs`.

If the optional `readonly` flag is set to `true`, then writes to the filesystem return error `EROFS`.

`options` is a comma-separated list of mount options. See **guestmount**(1) for some useful options.

`cachetimeout` sets the timeout (in seconds) for cached directory entries. The default is 60 seconds. See **guestmount**(1) for further information.

If `debugcalls` is set to `true`, then additional debugging information is generated for every FUSE call.

When “`mount-local`” returns, the filesystem is ready, but is not processing requests (access to it will block). You have to call “`mount-local-run`” to run the main loop.

See “`MOUNT LOCAL`” in **guestfs**(3) for full documentation.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

**mount-local-run**

```
mount-local-run
```

Run the main loop which translates kernel calls to libguestfs calls.

This should only be called after “mount-local” returns successfully. The call will not return until the filesystem is unmounted.

**Note** you must *not* make concurrent libguestfs calls on the same handle from another thread.

You may call this from a different thread than the one which called “mount-local”, subject to the usual rules for threads and libguestfs (see “MULTIPLE HANDLES AND MULTIPLE THREADS” in **guestfs**(3)).

See “MOUNT LOCAL” in **guestfs**(3) for full documentation.

**mount-loop**

```
mount-loop file mountpoint
```

This command lets you mount *file* (a filesystem image in a file) on a mount point. It is entirely equivalent to the command `mount -o loop file mountpoint`.

**mount-options**

```
mount-options options mountable mountpoint
```

This is the same as the “mount” command, but it allows you to set the mount options as for the **mount**(8) `-o` flag.

If the `options` parameter is an empty string, then no options are passed (all options default to whatever the filesystem uses).

**mount-ro**

```
mount-ro mountable mountpoint
```

This is the same as the “mount” command, but it mounts the filesystem with the read-only (`-o ro`) flag.

**mount-vfs**

```
mount-vfs options vfstype mountable mountpoint
```

This is the same as the “mount” command, but it allows you to set both the mount options and the vfstype as for the **mount**(8) `-o` and `-t` flags.

**mountable-device**

```
mountable-device mountable
```

Returns the device name of a mountable. In quite a lot of cases, the mountable is the device name.

However this doesn’t apply for btrfs subvolumes, where the mountable is a combination of both the device name and the subvolume path (see also “mountable-subvolume” to extract the subvolume path of the mountable if any).

**mountable-subvolume**

```
mountable-subvolume mountable
```

Returns the subvolume path of a mountable. Btrfs subvolumes mountables are a combination of both the device name and the subvolume path (see also “mountable-device” to extract the device of the mountable).

If the mountable does not represent a btrfs subvolume, then this function fails and the `errno` is set to `EINVAL`.

**mountpoints**

```
mountpoints
```

This call is similar to “mounts”. That call returns a list of devices. This one returns a hash table (map) of device name to directory where the device is mounted.

**mounts**

```
mounts
```

This returns the list of currently mounted filesystems. It returns the list of devices (eg. `/dev/sda1`,

`/dev/VG/LV`).

Some internal mounts are not shown.

See also: “mountpoints”

## **mv**

```
mv src dest
```

This moves a file from `src` to `dest` where `dest` is either a destination filename or destination directory.

See also: “rename”.

## **nr-devices**

```
nr-devices
```

This returns the number of whole block devices that were added. This is the same as the number of devices that would be returned if you called “list-devices”.

To find out the maximum number of devices that could be added, call “max-disks”.

## **ntfs-3g-probe**

```
ntfs-3g-probe true|false device
```

This command runs the **ntfs-3g.probe** (8) command which probes an NTFS device for mountability. (Not all NTFS volumes can be mounted read-write, and some cannot be mounted at all).

`rw` is a boolean flag. Set it to `true` if you want to test if the volume can be mounted read-write. Set it to `false` if you want to test if the volume can be mounted read-only.

The return value is an integer which 0 if the operation would succeed, or some non-zero value documented in the **ntfs-3g.probe** (8) manual page.

This command depends on the feature `ntfs3g`. See also “feature-available”.

## **ntfscat-i**

```
ntfscat-i device inode (filename|-)
```

Download a file given its inode from a NTFS filesystem and save it as *filename* on the local machine.

This allows to download some otherwise inaccessible files such as the ones within the `$Extend` folder.

The filesystem from which to extract the file must be unmounted, otherwise the call will fail.

Use `-` instead of a filename to read/write from stdin/stdout.

## **ntfsclose-in**

```
ntfsclose-in (backupfile|-) device
```

Restore the `backupfile` (from a previous call to “ntfsclose-out”) to `device`, overwriting any existing contents of this device.

Use `-` instead of a filename to read/write from stdin/stdout.

This command depends on the feature `ntfs3g`. See also “feature-available”.

## **ntfsclose-out**

```
ntfsclose-out device (backupfile|-) [metadataonly:true|false] [rescue:true|false]
```

Stream the NTFS filesystem `device` to the local file `backupfile`. The format used for the backup file is a special format used by the **ntfsclose** (8) tool.

If the optional `metadataonly` flag is `true`, then *only* the metadata is saved, losing all the user data (this is useful for diagnosing some filesystem problems).

The optional `rescue`, `ignorefscheck`, `preservetimestamps` and `force` flags have precise meanings detailed in the **ntfsclose** (8) man page.

Use “ntfsclose-in” to restore the file back to a libguestfs device.

Use `-` instead of a filename to read/write from stdin/stdout.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

This command depends on the feature `ntfs3g`. See also “feature-available”.

### **ntfsfix**

```
ntfsfix device [clearbadsectors:true|false]
```

This command repairs some fundamental NTFS inconsistencies, resets the NTFS journal file, and schedules an NTFS consistency check for the first boot into Windows.

This is *not* an equivalent of Windows `chkdsk`. It does*not* scan the filesystem for inconsistencies.

The optional `clearbadsectors` flag clears the list of bad sectors. This is useful after cloning a disk with bad sectors to a new disk.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

This command depends on the feature `ntfs3g`. See also “feature-available”.

### **ntfsresize**

#### **ntfsresize-opts**

```
ntfsresize device [size:N] [force:true|false]
```

This command resizes an NTFS filesystem, expanding or shrinking it to the size of the underlying device.

The optional parameters are:

#### **size**

The new size (in bytes) of the filesystem. If omitted, the filesystem is resized to fit the container (eg. partition).

#### **force**

If this option is true, then force the resize of the filesystem even if the filesystem is marked as requiring a consistency check.

After the resize operation, the filesystem is always marked as requiring a consistency check (for safety). You have to boot into Windows to perform this check and clear this condition. If you *don't* set the `force` option then it is not possible to call “ntfsresize” multiple times on a single filesystem without booting into Windows between each resize.

See also **ntfsresize** (8).

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

This command depends on the feature `ntfsprogs`. See also “feature-available”.

#### **ntfsresize-size**

```
ntfsresize-size device size
```

This command is the same as “ntfsresize” except that it allows you to specify the new size (in bytes) explicitly.

*This function is deprecated.* In new code, use the “ntfsresize” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

This command depends on the feature `ntfsprogs`. See also “feature-available”.

### **parse-environment**

```
parse-environment
```

Parse the program's environment and set flags in the handle accordingly. For example if `LIBGUESTFS_DEBUG=1` then the ‘verbose’ flag is set in the handle.

*Most programs do not need to call this.* It is done implicitly when you call “create”.

See “ENVIRONMENT VARIABLES” in **guestfs** (3) for a list of environment variables that can affect libguestfs handles. See also “guestfs\_create\_flags” in **guestfs** (3), and “parse-environment-list”.



**parse-environment-list**

```
parse-environment-list 'environment ...'
```

Parse the list of strings in the argument `environment` and set flags in the handle accordingly. For example if `LIBGUESTFS_DEBUG=1` is a string in the list, then the ‘verbose’ flag is set in the handle.

This is the same as “`parse-environment`” except that it parses an explicit list of strings instead of the program’s environment.

**part-add**

```
part-add device prlogex startsect endsect
```

This command adds a partition to device. If there is no partition table on the device, call “`part-init`” first.

The `prlogex` parameter is the type of partition. Normally you should pass `p` or `primary` here, but MBR partition tables also support `l` (or `logical`) and `e` (or `extended`) partition types.

`startsect` and `endsect` are the start and end of the partition in *sectors*. `endsect` may be negative, which means it counts backwards from the end of the disk (`-1` is the last sector).

Creating a partition which covers the whole disk is not so easy. Use “`part-disk`” to do that.

**part-del**

```
part-del device partnum
```

This command deletes the partition numbered `partnum` on device.

Note that in the case of MBR partitioning, deleting an extended partition also deletes any logical partitions it contains.

**part-disk**

```
part-disk device parttype
```

This command is simply a combination of “`part-init`” followed by “`part-add`” to create a single primary partition covering the whole disk.

`parttype` is the partition table type, usually `mbr` or `gpt`, but other possible values are described in “`part-init`”.

**part-expand-gpt**

```
part-expand-gpt device
```

Move backup GPT data structures to the end of the disk. This is useful in case of in-place image expand since disk space after backup GPT header is not usable. This is equivalent to `sgdisk -e`.

See also **sgdisk** (8).

This command depends on the feature `gdisk`. See also “`feature-available`”.

**part-get-bootable**

```
part-get-bootable device partnum
```

This command returns true if the partition `partnum` on device has the bootable flag set.

See also “`part-set-bootable`”.

**part-get-disk-guid**

```
part-get-disk-guid device
```

Return the disk identifier (GUID) of a GPT-partitioned device. Behaviour is undefined for other partition types.

This command depends on the feature `gdisk`. See also “`feature-available`”.

**part-get-gpt-attributes**

```
part-get-gpt-attributes device partnum
```

Return the attribute flags of numbered GPT partition `partnum`. An error is returned for MBR partitions.

This command depends on the feature `gdisk`. See also “`feature-available`”.

**part-get-gpt-guid**

```
part-get-gpt-guid device partnum
```

Return the GUID of numbered GPT partition `partnum`.

This command depends on the feature `gdisk`. See also “feature-available”.

**part-get-gpt-type**

```
part-get-gpt-type device partnum
```

Return the type GUID of numbered GPT partition `partnum`. For MBR partitions, return an appropriate GUID corresponding to the MBR type. Behaviour is undefined for other partition types.

This command depends on the feature `gdisk`. See also “feature-available”.

**part-get-mbr-id**

```
part-get-mbr-id device partnum
```

Returns the MBR type byte (also known as the ID byte) from the numbered partition `partnum`.

Note that only MBR (old DOS-style) partitions have type bytes. You will get undefined results for other partition table types (see “part-get-parttype”).

**part-get-mbr-part-type**

```
part-get-mbr-part-type device partnum
```

This returns the partition type of an MBR partition numbered `partnum` on device `device`.

It returns `primary`, `logical`, or `extended`.

**part-get-name**

```
part-get-name device partnum
```

This gets the partition name on partition numbered `partnum` on device `device`. Note that partitions are numbered from 1.

The partition name can only be read on certain types of partition table. This works on `gpt` but not on `mbr` partitions.

**part-get-parttype**

```
part-get-parttype device
```

This command examines the partition table on `device` and returns the partition table type (format) being used.

Common return values include: `msdos` (a DOS/Windows style MBR partition table), `gpt` (a GPT/EFI-style partition table). Other values are possible, although unusual. See “part-init” for a full list.

**part-init**

```
part-init device parttype
```

This creates an empty partition table on `device` of one of the partition types listed below. Usually `parttype` should be either `msdos` or `gpt` (for large disks).

Initially there are no partitions. Following this, you should call “part-add” for each partition required.

Possible values for `parttype` are:

`efi`

`gpt`

Intel EFI / GPT partition table.

This is recommended for  $\geq 2$  TB partitions that will be accessed from Linux and Intel-based Mac OS X. It also has limited backwards compatibility with the `mbr` format.

`mbr`

`msdos`

The standard PC “Master Boot Record” (MBR) format used by MS-DOS and Windows. This partition type will **only** work for device sizes up to 2 TB. For large disks we recommend using `gpt`.

Other partition table types that may work but are not supported include:

`aix`  
AIX disk labels.

`amiga`  
`rdb`  
Amiga “Rigid Disk Block” format.

`bsd`  
BSD disk labels.

`dasd`  
DASD, used on IBM mainframes.

`dvh`  
MIPS/SGI volumes.

`mac`  
Old Mac partition format. Modern Macs use `gpt`.

`pc98`  
NEC PC-98 format, common in Japan apparently.

`sun`  
Sun disk labels.

#### **part-list**

`part-list device`

This command parses the partition table on `device` and returns the list of partitions found.

The fields in the returned structure are:

`part_num`  
Partition number, counting from 1.

`part_start`  
Start of the partition *in bytes*. To get sectors you have to divide by the device’s sector size, see “`blockdev-getss`”.

`part_end`  
End of the partition in bytes.

`part_size`  
Size of the partition in bytes.

#### **part-resize**

`part-resize device partnum endsect`

This command resizes the partition numbered `partnum` on `device` by moving the end position.

Note that this does not modify any filesystem present in the partition. If you wish to do this, you will need to use filesystem resizing commands like “`resize2fs`”.

When growing a partition you will want to grow the filesystem afterwards, but when shrinking, you need to shrink the filesystem before the partition.

#### **part-set-bootable**

`part-set-bootable device partnum true|false`

This sets the bootable flag on partition numbered `partnum` on `device`. Note that partitions are numbered from 1.

The bootable flag is used by some operating systems (notably Windows) to determine which partition to boot from. It is by no means universally recognized.

**part-set-disk-guid**

```
part-set-disk-guid device guid
```

Set the disk identifier (GUID) of a GPT-partitioned device to `guid`. Return an error if the partition table of device isn't GPT, or if `guid` is not a valid GUID.

This command depends on the feature `gdisk`. See also “feature-available”.

**part-set-disk-guid-random**

```
part-set-disk-guid-random device
```

Set the disk identifier (GUID) of a GPT-partitioned device to a randomly generated value. Return an error if the partition table of device isn't GPT.

This command depends on the feature `gdisk`. See also “feature-available”.

**part-set-gpt-attributes**

```
part-set-gpt-attributes device partnum attributes
```

Set the attribute flags of numbered GPT partition `partnum` to `attributes`. Return an error if the partition table of device isn't GPT.

See [https://en.wikipedia.org/wiki/GUID\\_Partition\\_Table#Partition\\_entries](https://en.wikipedia.org/wiki/GUID_Partition_Table#Partition_entries) for a useful list of partition attributes.

This command depends on the feature `gdisk`. See also “feature-available”.

**part-set-gpt-guid**

```
part-set-gpt-guid device partnum guid
```

Set the GUID of numbered GPT partition `partnum` to `guid`. Return an error if the partition table of device isn't GPT, or if `guid` is not a valid GUID.

This command depends on the feature `gdisk`. See also “feature-available”.

**part-set-gpt-type**

```
part-set-gpt-type device partnum guid
```

Set the type GUID of numbered GPT partition `partnum` to `guid`. Return an error if the partition table of device isn't GPT, or if `guid` is not a valid GUID.

See [https://en.wikipedia.org/wiki/GUID\\_Partition\\_Table#Partition\\_type\\_GUIDs](https://en.wikipedia.org/wiki/GUID_Partition_Table#Partition_type_GUIDs) for a useful list of type GUIDs.

This command depends on the feature `gdisk`. See also “feature-available”.

**part-set-mbr-id**

```
part-set-mbr-id device partnum idbyte
```

Sets the MBR type byte (also known as the ID byte) of the numbered partition `partnum` to `idbyte`. Note that the type bytes quoted in most documentation are in fact hexadecimal numbers, but usually documented without any leading “0x” which might be confusing.

Note that only MBR (old DOS-style) partitions have type bytes. You will get undefined results for other partition table types (see “part-get-parttype”).

**part-set-name**

```
part-set-name device partnum name
```

This sets the partition name on partition numbered `partnum` on device `device`. Note that partitions are numbered from 1.

The partition name can only be set on certain types of partition table. This works on `gpt` but not on `mbr` partitions.

**part-to-dev**

```
part-to-dev partition
```

This function takes a partition name (eg. “/dev/sdb1”) and removes the partition number, returning the

device name (eg. “/dev/sdb”).

The named partition must exist, for example as a string returned from “list-partitions”.

See also “part-to-partnum”, “device-index”.

### **part-to-partnum**

`part-to-partnum partition`

This function takes a partition name (eg. “/dev/sdb1”) and returns the partition number (eg. 1).

The named partition must exist, for example as a string returned from “list-partitions”.

See also “part-to-dev”.

### **ping-daemon**

`ping-daemon`

This is a test probe into the guestfs daemon running inside the libguestfs appliance. Calling this function checks that the daemon responds to the ping message, without affecting the daemon or attached block device(s) in any other way.

### **pread**

`pread path count offset`

This command lets you read part of a file. It reads `count` bytes of the file, starting at `offset`, from file `path`.

This may read fewer bytes than requested. For further details see the **pread** (2) system call.

See also “pwrite”, “pread-device”.

Because of the message protocol, there is a transfer limit of somewhere between 2MB and 4MB. See “PROTOCOL LIMITS” in **guestfs** (3).

### **pread-device**

`pread-device device count offset`

This command lets you read part of a block device. It reads `count` bytes of device, starting at `offset`.

This may read fewer bytes than requested. For further details see the **pread** (2) system call.

See also “pread”.

Because of the message protocol, there is a transfer limit of somewhere between 2MB and 4MB. See “PROTOCOL LIMITS” in **guestfs** (3).

### **pvchange-uuid**

`pvchange-uuid device`

Generate a new random UUID for the physical volume device.

This command depends on the feature `lvm2`. See also “feature-available”.

### **pvchange-uuid-all**

`pvchange-uuid-all`

Generate new random UUIDs for all physical volumes.

This command depends on the feature `lvm2`. See also “feature-available”.

### **pvcreate**

`pvcreate device`

This creates an LVM physical volume on the named `device`, where `device` should usually be a partition name such as `/dev/sda1`.

This command depends on the feature `lvm2`. See also “feature-available”.

**pvremove**

pvremove device

This wipes a physical volume device so that LVM will no longer recognise it.

The implementation uses the **pvremove**(8) command which refuses to wipe physical volumes that contain any volume groups, so you have to remove those first.

This command depends on the feature lvm2. See also “feature-available”.

**pvresize**

pvresize device

This resizes (expands or shrinks) an existing LVM physical volume to match the new size of the underlying device.

This command depends on the feature lvm2. See also “feature-available”.

**pvresize-size**

pvresize-size device size

This command is the same as “pvresize” except that it allows you to specify the new size (in bytes) explicitly.

This command depends on the feature lvm2. See also “feature-available”.

**pvs**

pvs

List all the physical volumes detected. This is the equivalent of the **pvs**(8) command.

This returns a list of just the device names that contain PVs (eg. */dev/sda2*).

See also “pvs-full”.

This command depends on the feature lvm2. See also “feature-available”.

**pvs-full**

pvs-full

List all the physical volumes detected. This is the equivalent of the **pvs**(8) command. The “full” version includes all fields.

This command depends on the feature lvm2. See also “feature-available”.

**pvuuid**

pvuuid device

This command returns the UUID of the LVM PV device.

**pwrite**

pwrite path content offset

This command writes to part of a file. It writes the data buffer content to the file path starting at offset offset.

This command implements the **pwrite**(2) system call, and like that system call it may not write the full data requested. The return value is the number of bytes that were actually written to the file. This could even be 0, although short writes are unlikely for regular files in ordinary circumstances.

See also “pread”, “pwrite-device”.

Because of the message protocol, there is a transfer limit of somewhere between 2MB and 4MB. See “PROTOCOL LIMITS” in **guestfs**(3).

**pwrite-device**

pwrite-device device content offset

This command writes to part of a device. It writes the data buffer content to device starting at offset offset.

This command implements the **pwrite**(2) system call, and like that system call it may not write the full data requested (although short writes to disk devices and partitions are probably impossible with standard Linux kernels).

See also “pwrite”.

Because of the message protocol, there is a transfer limit of somewhere between 2MB and 4MB. See “PROTOCOL LIMITS” in **guestfs**(3).

### **read-file**

```
read-file path
```

This call returns the contents of the file `path` as a buffer.

Unlike “cat”, this function can correctly handle files that contain embedded ASCII NUL characters.

### **read-lines**

```
read-lines path
```

Return the contents of the file named `path`.

The file contents are returned as a list of lines. Trailing LF and CRLF character sequences are *not* returned.

Note that this function cannot correctly handle binary files (specifically, files containing \0 character which is treated as end of string). For those you need to use the “read-file” function and split the buffer into lines yourself.

### **readdir**

```
readdir dir
```

This returns the list of directory entries in directory `dir`.

All entries in the directory are returned, including `.` and `...`. The entries are *not* sorted, but returned in the same order as the underlying filesystem.

Also this call returns basic file type information about each file. The `ftype` field will contain one of the following characters:

`'b'` Block special

`'c'` Char special

`'d'` Directory

`'f'` FIFO (named pipe)

`'l'` Symbolic link

`'r'` Regular file

`'s'` Socket

`'u'` Unknown file type

`'?'` The **readdir**(3) call returned a `d_type` field with an unexpected value

This function is primarily intended for use by programs. To get a simple list of names, use “ls”. To get a printable directory for human consumption, use “ll”.

Because of the message protocol, there is a transfer limit of somewhere between 2MB and 4MB. See “PROTOCOL LIMITS” in **guestfs**(3).

### **readlink**

```
readlink path
```

This command reads the target of a symbolic link.

### **readlinklist**

```
readlinklist path 'names ...'
```

This call allows you to do a `readlink` operation on multiple files, where all files are in the directory

`path`. `names` is the list of files from this directory.

On return you get a list of strings, with a one-to-one correspondence to the `names` list. Each string is the value of the symbolic link.

If the **readlink**(2) operation fails on any name, then the corresponding result string is the empty string `" "`. However the whole operation is completed even if there were **readlink**(2) errors, and so you can call this function with names where you don't know if they are symbolic links already (albeit slightly less efficient).

This call is intended for programs that want to efficiently list a directory contents without making many round-trips.

### **realpath**

```
realpath path
```

Return the canonicalized absolute pathname of `path`. The returned path has no `.`, `..` or symbolic link path elements.

### **remount**

```
remount mountpoint [rw:true|false]
```

This call allows you to change the `rw` (readonly/read-write) flag on an already mounted filesystem at `mountpoint`, converting a readonly filesystem to be read-write, or vice-versa.

Note that at the moment you must supply the “optional” `rw` parameter. In future we may allow other flags to be adjusted.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

### **remove-drive**

```
remove-drive label
```

This function is conceptually the opposite of “add-drive-opts”. It removes the drive that was previously added with label `label`.

Note that in order to remove drives, you have to add them with labels (see the optional `label` argument to “add-drive-opts”). If you didn't use a label, then they cannot be removed.

You can call this function before or after launching the handle. If called after launch, if the backend supports it, we try to hot unplug the drive: see “HOTPLUGGING” in **guestfs**(3). The disk **must not** be in use (eg. mounted) when you do this. We try to detect if the disk is in use and stop you from doing this.

### **removexattr**

```
removexattr xattr path
```

This call removes the extended attribute named `xattr` of the file `path`.

See also: “lremovexattr”, **attr**(5).

This command depends on the feature `linuxxattrs`. See also “feature-available”.

### **rename**

```
rename oldpath newpath
```

Rename a file to a new place on the same filesystem. This is the same as the Linux **rename**(2) system call. In most cases you are better to use “mv” instead.

### **resize2fs**

```
resize2fs device
```

This resizes an ext2, ext3 or ext4 filesystem to match the size of the underlying device.

See also “RESIZE2FS ERRORS” in **guestfs**(3).

### **resize2fs-M**

```
resize2fs-M device
```

This command is the same as “resize2fs”, but the filesystem is resized to its minimum size. This works like the `-M` option to the **resize2fs**(8) command.



To get the resulting size of the filesystem you should call “tune2fs-l” and read the `Block size` and `Block count` values. These two numbers, multiplied together, give the resulting size of the minimal filesystem in bytes.

See also “RESIZE2FS ERRORS” in **guestfs** (3).

### **resize2fs-size**

```
resize2fs-size device size
```

This command is the same as “resize2fs” except that it allows you to specify the new size (in bytes) explicitly.

See also “RESIZE2FS ERRORS” in **guestfs** (3).

### **rm**

```
rm path
```

Remove the single file `path`.

### **rm-f**

```
rm-f path
```

Remove the file `path`.

If the file doesn’t exist, that error is ignored. (Other errors, eg. I/O errors or bad paths, are not ignored)

This call cannot remove directories. Use “rmdir” to remove an empty directory, or “rm-rf” to remove directories recursively.

### **rm-rf**

```
rm-rf path
```

Remove the file or directory `path`, recursively removing the contents if its a directory. This is like the `rm -rf` shell command.

### **rmdir**

```
rmdir path
```

Remove the single directory `path`.

### **rmmountpoint**

```
rmmountpoint exemptpath
```

This call removes a mountpoint that was previously created with “mkmountpoint”. See “mkmountpoint” for full details.

### **rsync**

```
rsync src dest [archive:true|false] [deletedest:true|false]
```

This call may be used to copy or synchronize two directories under the same libguestfs handle. This uses the **rsync**(1) program which uses a fast algorithm that avoids copying files unnecessarily.

`src` and `dest` are the source and destination directories. Files are copied from `src` to `dest`.

The optional arguments are:

**archive**

Turns on archive mode. This is the same as passing the `--archive` flag to `rsync`.

**deletedest**

Delete files at the destination that do not exist at the source.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

This command depends on the feature `rsync`. See also “feature-available”.

### **rsync-in**

```
rsync-in remote dest [archive:true|false] [deletedest:true|false]
```

This call may be used to copy or synchronize the filesystem on the host or on a remote computer with the

filesystem within libguestfs. This uses the **rsync**(1) program which uses a fast algorithm that avoids copying files unnecessarily.

This call only works if the network is enabled. See “set-network” or the `--network` option to various tools like **guestfish**(1).

Files are copied from the remote server and directory specified by `remote` to the destination directory `dest`.

The format of the remote server string is defined by **rsync**(1). Note that there is no way to supply a password or passphrase so the target must be set up not to require one.

The optional arguments are the same as those of “rsync”.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

This command depends on the feature `rsync`. See also “feature-available”.

### **rsync-out**

```
rsync-out src remote [archive:true|false] [deletedest:true|false]
```

This call may be used to copy or synchronize the filesystem within libguestfs with a filesystem on the host or on a remote computer. This uses the **rsync**(1) program which uses a fast algorithm that avoids copying files unnecessarily.

This call only works if the network is enabled. See “set-network” or the `--network` option to various tools like **guestfish**(1).

Files are copied from the source directory `src` to the remote server and directory specified by `remote`.

The format of the remote server string is defined by **rsync**(1). Note that there is no way to supply a password or passphrase so the target must be set up not to require one.

The optional arguments are the same as those of “rsync”.

Globbing does not happen on the `src` parameter. In programs which use the API directly you have to expand wildcards yourself (see “glob-expand”). In **guestfish** you can use the `glob` command (see “glob”), for example:

```
><fs> glob rsync-out /* rsync://remote/
```

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

This command depends on the feature `rsync`. See also “feature-available”.

### **scrub-device**

```
scrub-device device
```

This command writes patterns over `device` to make data retrieval more difficult.

It is an interface to the **scrub**(1) program. See that manual page for more details.

This command depends on the feature `scrub`. See also “feature-available”.

### **scrub-file**

```
scrub-file file
```

This command writes patterns over a file to make data retrieval more difficult.

The file is *removed* after scrubbing.

It is an interface to the **scrub**(1) program. See that manual page for more details.

This command depends on the feature `scrub`. See also “feature-available”.

### **scrub-freespace**

```
scrub-freespace dir
```

This command creates the directory `dir` and then fills it with files until the filesystem is full, and scrubs the files as for “scrub-file”, and deletes them. The intention is to scrub any free space on the partition containing `dir`.

It is an interface to the **scrub**(1) program. See that manual page for more details.

This command depends on the feature `scrub`. See also “feature-available”.

### **selinux-relabel**

```
selinux-relabel specfile path [force:true|false]
```

SELinux relabel parts of the filesystem.

The `specfile` parameter controls the policy spec file used. You have to parse `/etc/selinux/config` to find the correct SELinux policy and then pass the spec file, usually: `/etc/selinux/ + selinuxtype + /contexts/files/file_contexts`.

The required `path` parameter is the top level directory where relabelling starts. Normally you should pass `path` as `/` to relabel the whole guest filesystem.

The optional `force` boolean controls whether the context is reset for customizable files, and also whether the user, role and range parts of the file context is changed.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

This command depends on the feature `selinuxrelabel`. See also “feature-available”.

### **set-append**

#### **append**

```
set-append append
```

This function is used to add additional options to the libguestfs appliance kernel command line.

The default is `NULL` unless overridden by setting `LIBGUESTFS_APPEND` environment variable.

Setting `append` to `NULL` means *no* additional options are passed (libguestfs always adds a few of its own).

### **set-attach-method**

#### **attach-method**

```
set-attach-method backend
```

Set the method that libguestfs uses to connect to the backend `guestfsd` daemon.

See “BACKEND” in **guestfs**(3).

*This function is deprecated.* In new code, use the “set-backend” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### **set-autosync**

#### **autosync**

```
set-autosync true|false
```

If `autosync` is `true`, this enables `autosync`. Libguestfs will make a best effort attempt to make filesystems consistent and synchronized when the handle is closed (also if the program exits without closing handles).

This is enabled by default (since libguestfs 1.5.24, previously it was disabled by default).

### **set-backend**

#### **backend**

```
set-backend backend
```

Set the method that libguestfs uses to connect to the backend `guestfsd` daemon.

This handle property was previously called the “attach method”.

See “BACKEND” in **guestfs**(3).

### **set-backend-setting**

```
set-backend-setting name val
```

Append “`name=value`” to the backend settings string list. However if a string already exists matching “`name`” or beginning with “`name=`”, then that setting is replaced.

See “BACKEND” in **guestfs**(3), “BACKEND SETTINGS” in **guestfs**(3).

### **set-backend-settings**

```
set-backend-settings 'settings ...'
```

Set a list of zero or more settings which are passed through to the current backend. Each setting is a string which is interpreted in a backend-specific way, or ignored if not understood by the backend.

The default value is an empty list, unless the environment variable `LIBGUESTFS_BACKEND_SETTINGS` was set when the handle was created. This environment variable contains a colon-separated list of settings.

This call replaces all backend settings. If you want to replace a single backend setting, see “set-backend-setting”. If you want to clear a single backend setting, see “clear-backend-setting”.

See “BACKEND” in **guestfs**(3), “BACKEND SETTINGS” in **guestfs**(3).

### **set-cachedir**

#### **cachedir**

```
set-cachedir cachedir
```

Set the directory used by the handle to store the appliance cache, when using a supermin appliance. The appliance is cached and shared between all handles which have the same effective user ID.

The environment variables `LIBGUESTFS_CACHEDIR` and `TMPDIR` control the default value: If `LIBGUESTFS_CACHEDIR` is set, then that is the default. Else if `TMPDIR` is set, then that is the default. Else `/var/tmp` is the default.

### **set-direct**

#### **direct**

```
set-direct true|false
```

If the direct appliance mode flag is enabled, then stdin and stdout are passed directly through to the appliance once it is launched.

One consequence of this is that log messages aren’t caught by the library and handled by “set-log-message-callback”, but go straight to stdout.

You probably don’t want to use this unless you know what you are doing.

The default is disabled.

*This function is deprecated.* In new code, use the “internal-get-console-socket” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### **set-e2attrs**

```
set-e2attrs file attrs [clear:true|false]
```

This sets or clears the file attributes `attrs` associated with the inode *file*.

`attrs` is a string of characters representing file attributes. See “get-e2attrs” for a list of possible attributes. Not all attributes can be changed.

If optional boolean `clear` is not present or false, then the `attrs` listed are set in the inode.

If `clear` is true, then the `attrs` listed are cleared in the inode.

In both cases, other attributes not present in the `attrs` string are left unchanged.

These attributes are only present when the file is located on an ext2/3/4 filesystem. Using this call on other filesystem types will result in an error.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

### **set-e2generation**

```
set-e2generation file generation
```

This sets the ext2 file generation of a file.

See “get-e2generation”.

### **set-e2label**

```
set-e2label device label
```

This sets the ext2/3/4 filesystem label of the filesystem on `device` to `label`. Filesystem labels are limited to 16 characters.

You can use either “tune2fs-l” or “get-e2label” to return the existing label on a filesystem.

*This function is deprecated.* In new code, use the “set-label” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### **set-e2uuid**

```
set-e2uuid device uuid
```

This sets the ext2/3/4 filesystem UUID of the filesystem on `device` to `uuid`. The format of the UUID and alternatives such as `clear`, `random` and `time` are described in the **tune2fs** (8) manpage.

You can use “vfs-uuid” to return the existing UUID of a filesystem.

*This function is deprecated.* In new code, use the “set-uuid” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### **set-hv**

#### **hv**

```
set-hv hv
```

Set the hypervisor binary that we will use. The hypervisor depends on the backend, but is usually the location of the qemu/KVM hypervisor. For the uml backend, it is the location of the `linux` or `vmlinux` binary.

The default is chosen when the library was compiled by the configure script.

You can also override this by setting the `LIBGUESTFS_HV` environment variable.

Note that you should call this function as early as possible after creating the handle. This is because some pre-launch operations depend on testing qemu features (by running `qemu -help`). If the qemu binary changes, we don't retest features, and so you might see inconsistent results. Using the environment variable `LIBGUESTFS_HV` is safest of all since that picks the qemu binary at the same time as the handle is created.

### **set-identifier**

#### **identifier**

```
set-identifier identifier
```

This is an informative string which the caller may optionally set in the handle. It is printed in various places, allowing the current handle to be identified in debugging output.

One important place is when tracing is enabled. If the identifier string is not an empty string, then trace messages change from this:

```
libguestfs: trace: get_tmpdir
libguestfs: trace: get_tmpdir = "/tmp"
```

to this:

```
libguestfs: trace: ID: get_tmpdir
libguestfs: trace: ID: get_tmpdir = "/tmp"
```

where `ID` is the identifier string set by this call.

The identifier must only contain alphanumeric ASCII characters, underscore and minus sign. The default is the empty string.

See also “set-program”, “set-trace”, “get-identifier”.

### **set-label**

```
set-label mountable label
```

Set the filesystem label on mountable to label.

Only some filesystem types support labels, and libguestfs supports setting labels on only a subset of these.

ext2, ext3, ext4

Labels are limited to 16 bytes.

NTFS

Labels are limited to 128 unicode characters.

XFS

The label is limited to 12 bytes. The filesystem must not be mounted when trying to set the label.

btrfs

The label is limited to 255 bytes and some characters are not allowed. Setting the label on a btrfs subvolume will set the label on its parent filesystem. The filesystem must not be mounted when trying to set the label.

fat The label is limited to 11 bytes.

swap

The label is limited to 16 bytes.

If there is no support for changing the label for the type of the specified filesystem, set\_label will fail and set errno as ENOTSUP.

To read the label on a filesystem, call “vfs-label”.

### **set-libvirt-requested-credential**

```
set-libvirt-requested-credential index cred
```

After requesting the index'th credential from the user, call this function to pass the answer back to libvirt.

See “LIBVIRT AUTHENTICATION” in **guestfs**(3) for documentation and example code.

### **set-libvirt-supported-credentials**

```
set-libvirt-supported-credentials 'creds ...'
```

Call this function before setting an event handler for GUESTFS\_EVENT\_LIBVIRT\_AUTH, to supply the list of credential types that the program knows how to process.

The creds list must be a non-empty list of strings. Possible strings are:

```
username
authname
language
cnonce
passphrase
echoprompt
noechoprompt
realm
external
```

See libvirt documentation for the meaning of these credential types.

See “LIBVIRT AUTHENTICATION” in **guestfs**(3) for documentation and example code.

### **set-memsize**

#### **memsize**

```
set-memsize memsize
```

This sets the memory size in megabytes allocated to the hypervisor. This only has any effect if called before “launch”.

You can also change this by setting the environment variable `LIBGUESTFS_MEMSIZE` before the handle is created.

For more information on the architecture of libguestfs, see **guestfs** (3).

### **set-network**

#### **network**

```
set-network true|false
```

If `network` is true, then the network is enabled in the libguestfs appliance. The default is false.

This affects whether commands are able to access the network (see “RUNNING COMMANDS” in **guestfs** (3)).

You must call this before calling “launch”, otherwise it has no effect.

### **set-path**

#### **path**

```
set-path searchpath
```

Set the path that libguestfs searches for kernel and initrd.img.

The default is `$libdir/guestfs` unless overridden by setting `LIBGUESTFS_PATH` environment variable.

Setting `path` to `NULL` restores the default path.

### **set-pgroup**

#### **pgroup**

```
set-pgroup true|false
```

If `pgroup` is true, child processes are placed into their own process group.

The practical upshot of this is that signals like `SIGINT` (from users pressing `^C`) won’t be received by the child process.

The default for this flag is false, because usually you want `^C` to kill the subprocess. Guestfish sets this flag to true when used interactively, so that `^C` can cancel long-running commands gracefully (see “user-cancel”).

### **set-program**

#### **program**

```
set-program program
```

Set the program name. This is an informative string which the main program may optionally set in the handle.

When the handle is created, the program name in the handle is set to the basename from `argv[0]`. The program name can never be `NULL`.

### **set-qemu**

#### **qemu**

```
set-qemu hv
```

Set the hypervisor binary (usually `qemu`) that we will use.

The default is chosen when the library was compiled by the configure script.

You can also override this by setting the `LIBGUESTFS_HV` environment variable.

Setting `hv` to `NULL` restores the default `qemu` binary.

Note that you should call this function as early as possible after creating the handle. This is because some pre-launch operations depend on testing `qemu` features (by running `qemu -help`). If the `qemu` binary changes, we don’t retest features, and so you might see inconsistent results. Using the environment variable `LIBGUESTFS_HV` is safest of all since that picks the `qemu` binary at the same time as the handle is created.

*This function is deprecated.* In new code, use the “set-hv” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### **set-recovery-proc**

#### **recovery-proc**

```
set-recovery-proc true|false
```

If this is called with the parameter `false` then “launch” does not create a recovery process. The purpose of the recovery process is to stop runaway hypervisor processes in the case where the main program aborts abruptly.

This only has any effect if called before “launch”, and the default is `true`.

About the only time when you would want to disable this is if the main process will fork itself into the background (“daemonize” itself). In this case the recovery process thinks that the main program has disappeared and so kills the hypervisor, which is not very helpful.

### **set-selinux**

#### **selinux**

```
set-selinux true|false
```

This sets the `selinux` flag that is passed to the appliance at boot time. The default is `selinux=0` (disabled).

Note that if SELinux is enabled, it is always in Permissive mode (`enforcing=0`).

For more information on the architecture of `libguestfs`, see **guestfs**(3).

*This function is deprecated.* In new code, use the “selinux-relabel” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### **set-smp**

#### **smp**

```
set-smp smp
```

Change the number of virtual CPUs assigned to the appliance. The default is 1. Increasing this may improve performance, though often it has no effect.

This function must be called before “launch”.

### **set-tmpdir**

#### **tmpdir**

```
set-tmpdir tmpdir
```

Set the directory used by the handle to store temporary files.

The environment variables `LIBGUESTFS_TMPDIR` and `TMPDIR` control the default value: If `LIBGUESTFS_TMPDIR` is set, then that is the default. Else if `TMPDIR` is set, then that is the default. Else `/tmp` is the default.

### **set-trace**

#### **trace**

```
set-trace true|false
```

If the command `trace` flag is set to 1, then `libguestfs` calls, parameters and return values are traced.

If you want to trace C API calls into `libguestfs` (and other libraries) then possibly a better way is to use the external **ltrace**(1) command.

Command traces are disabled unless the environment variable `LIBGUESTFS_TRACE` is defined and set to 1.

Trace messages are normally sent to `stderr`, unless you register a callback to send them somewhere else (see “set-event-callback”).



**set-uuid**

```
set-uuid device uuid
```

Set the filesystem UUID on `device` to `uuid`. If this fails and the `errno` is `ENOTSUP`, means that there is no support for changing the UUID for the type of the specified filesystem.

Only some filesystem types support setting UUIDs.

To read the UUID on a filesystem, call “`vfs-uuid`”.

**set-uuid-random**

```
set-uuid-random device
```

Set the filesystem UUID on `device` to a random UUID. If this fails and the `errno` is `ENOTSUP`, means that there is no support for changing the UUID for the type of the specified filesystem.

Only some filesystem types support setting UUIDs.

To read the UUID on a filesystem, call “`vfs-uuid`”.

**set-verbose****verbose**

```
set-verbose true|false
```

If `verbose` is `true`, this turns on verbose messages.

Verbose messages are disabled unless the environment variable `LIBGUESTFS_DEBUG` is defined and set to 1.

Verbose messages are normally sent to `stderr`, unless you register a callback to send them somewhere else (see “`set-event-callback`”).

**setcon**

```
setcon context
```

This sets the SELinux security context of the daemon to the string `context`.

See the documentation about SELINUX in **guestfs** (3).

*This function is deprecated.* In new code, use the “`selinux-relabel`” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

This command depends on the feature `selinux`. See also “`feature-available`”.

**setxattr**

```
setxattr xattr val vallen path
```

This call sets the extended attribute named `xattr` of the file `path` to the value `val` (of length `vallen`). The value is arbitrary 8 bit data.

See also: “`lsetxattr`”, **attr** (5).

This command depends on the feature `linuxxattrs`. See also “`feature-available`”.

**sfdisk**

```
sfdisk device cyls heads sectors 'lines ...'
```

This is a direct interface to the **sfdisk** (8) program for creating partitions on block devices.

`device` should be a block device, for example `/dev/sda`.

`cyls`, `heads` and `sectors` are the number of cylinders, heads and sectors on the device, which are passed directly to **sfdisk** (8) as the `-C`, `-H` and `-S` parameters. If you pass 0 for any of these, then the corresponding parameter is omitted. Usually for ‘large’ disks, you can just pass 0 for these, but for small (floppy-sized) disks, **sfdisk** (8) (or rather, the kernel) cannot work out the right geometry and you will need to tell it.

`lines` is a list of lines that we feed to **sfdisk** (8). For more information refer to the **sfdisk** (8) manpage.

To create a single partition occupying the whole disk, you would pass `lines` as a single element list, when the single element being the string `,` (comma).

See also: “`sfdisk-l`”, “`sfdisk-N`”, “`part-init`”

*This function is deprecated.* In new code, use the “`part-add`” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### **sfdiskM**

```
sfdiskM device 'lines ...'
```

This is a simplified interface to the “`sfdisk`” command, where partition sizes are specified in megabytes only (rounded to the nearest cylinder) and you don’t need to specify the `cyls`, `heads` and `sectors` parameters which were rarely if ever used anyway.

See also: “`sfdisk`”, the **sfdisk** (8) manpage and “`part-disk`”

*This function is deprecated.* In new code, use the “`part-add`” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### **sfdisk-N**

```
sfdisk-N device partnum cyls heads sectors line
```

This runs **sfdisk** (8) option to modify just the single partition `n` (note: `n` counts from 1).

For other parameters, see “`sfdisk`”. You should usually pass 0 for the `cyls`/`heads`/`sectors` parameters.

See also: “`part-add`”

*This function is deprecated.* In new code, use the “`part-add`” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### **sfdisk-disk-geometry**

```
sfdisk-disk-geometry device
```

This displays the disk geometry of `device` read from the partition table. Especially in the case where the underlying block device has been resized, this can be different from the kernel’s idea of the geometry (see “`sfdisk-kernel-geometry`”).

The result is in human-readable format, and not designed to be parsed.

### **sfdisk-kernel-geometry**

```
sfdisk-kernel-geometry device
```

This displays the kernel’s idea of the geometry of `device`.

The result is in human-readable format, and not designed to be parsed.

### **sfdisk-l**

```
sfdisk-l device
```

This displays the partition table on `device`, in the human-readable output of the **sfdisk** (8) command. It is not intended to be parsed.

See also: “`part-list`”

*This function is deprecated.* In new code, use the “`part-list`” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### **sh**

```
sh command
```

This call runs a command from the guest filesystem via the guest’s `/bin/sh`.

This is like “command”, but passes the command to:

```
/bin/sh -c "command"
```

Depending on the guest’s shell, this usually results in wildcards being expanded, shell expressions being interpolated and so on.

All the provisos about “command” apply to this call.

### **sh-lines**

```
sh-lines command
```

This is the same as “sh”, but splits the result into a list of lines.

See also: “command-lines”

### **shutdown**

```
shutdown
```

This is the opposite of “launch”. It performs an orderly shutdown of the backend process(es). If the autosync flag is set (which is the default) then the disk image is synchronized.

If the subprocess exits with an error then this function will return an error, which should *not* be ignored (it may indicate that the disk image could not be written out properly).

It is safe to call this multiple times. Extra calls are ignored.

This call does *not* close or free up the handle. You still need to call “close” afterwards.

“close” will call this if you don’t do it explicitly, but note that any errors are ignored in that case.

### **sleep**

```
sleep secs
```

Sleep for *secs* seconds.

### **stat**

```
stat path
```

Returns file information for the given *path*.

This is the same as the **stat**(2) system call.

*This function is deprecated.* In new code, use the “statns” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### **statns**

```
statns path
```

Returns file information for the given *path*.

This is the same as the **stat**(2) system call.

### **statvfs**

```
statvfs path
```

Returns file system statistics for any mounted file system. *path* should be a file or directory in the mounted file system (typically it is the mount point itself, but it doesn’t need to be).

This is the same as the **statvfs**(2) system call.

### **strings**

```
strings path
```

This runs the **strings**(1) command on a file and returns the list of printable strings found.

The **strings** command has, in the past, had problems with parsing untrusted files. These are mitigated in the current version of libguestfs, but see “CVE-2014-8484” in **guestfs**(3).

Because of the message protocol, there is a transfer limit of somewhere between 2MB and 4MB. See

“PROTOCOL LIMITS” in **guestfs** (3).

### **strings-e**

`strings-e encoding path`

This is like the “strings” command, but allows you to specify the encoding of strings that are looked for in the source file `path`.

Allowed encodings are:

- s Single 7-bit-byte characters like ASCII and the ASCII-compatible parts of ISO-8859-X (this is what “strings” uses).
- S Single 8-bit-byte characters.
- b 16-bit big endian strings such as those encoded in UTF-16BE or UCS-2BE.

l (lower case letter L)

16-bit little endian such as UTF-16LE and UCS-2LE. This is useful for examining binaries in Windows guests.

B 32-bit big endian such as UCS-4BE.

L 32-bit little endian such as UCS-4LE.

The returned strings are transcoded to UTF-8.

The `strings` command has, in the past, had problems with parsing untrusted files. These are mitigated in the current version of `libguestfs`, but see “CVE-2014-8484” in **guestfs** (3).

Because of the message protocol, there is a transfer limit of somewhere between 2MB and 4MB. See “PROTOCOL LIMITS” in **guestfs** (3).

### **swapoff-device**

`swapoff-device device`

This command disables the `libguestfs` appliance swap device or partition named `device`. See “swapon-device”.

### **swapoff-file**

`swapoff-file file`

This command disables the `libguestfs` appliance swap on file.

### **swapoff-label**

`swapoff-label label`

This command disables the `libguestfs` appliance swap on labeled swap partition.

### **swapoff-uuid**

`swapoff-uuid uuid`

This command disables the `libguestfs` appliance swap partition with the given UUID.

This command depends on the feature `linuxfsuuid`. See also “feature-available”.

### **swapon-device**

`swapon-device device`

This command enables the `libguestfs` appliance to use the swap device or partition named `device`. The increased memory is made available for all commands, for example those run using “command” or “sh”.

Note that you should not swap to existing guest swap partitions unless you know what you are doing. They may contain hibernation information, or other information that the guest doesn’t want you to trash. You also risk leaking information about the host to the guest this way. Instead, attach a new host device to the guest and swap on that.

**swapon-file**

```
swapon-file file
```

This command enables swap to a file. See “swapon-device” for other notes.

**swapon-label**

```
swapon-label label
```

This command enables swap to a labeled swap partition. See “swapon-device” for other notes.

**swapon-uuid**

```
swapon-uuid uuid
```

This command enables swap to a swap partition with the given UUID. See “swapon-device” for other notes.

This command depends on the feature `linuxfsuuid`. See also “feature-available”.

**sync**

```
sync
```

This syncs the disk, so that any writes are flushed through to the underlying disk image.

You should always call this if you have modified a disk image, before closing the handle.

**syslinux**

```
syslinux device [directory:..]
```

Install the SYSLINUX bootloader on device.

The device parameter must be either a whole disk formatted as a FAT filesystem, or a partition formatted as a FAT filesystem. In the latter case, the partition should be marked as “active” (“part-set-bootable”) and a Master Boot Record must be installed (eg. using “pwrite-device”) on the first sector of the whole disk. The SYSLINUX package comes with some suitable Master Boot Records. See the **syslinux**(1) man page for further information.

The optional arguments are:

*directory*

Install SYSLINUX in the named subdirectory, instead of in the root directory of the FAT filesystem.

Additional configuration can be supplied to SYSLINUX by placing a file called *syslinux.cfg* on the FAT filesystem, either in the root directory, or under *directory* if that optional argument is being used. For further information about the contents of this file, see **syslinux**(1).

See also “extlinux”.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

This command depends on the feature `syslinux`. See also “feature-available”.

**tail**

```
tail path
```

This command returns up to the last 10 lines of a file as a list of strings.

Because of the message protocol, there is a transfer limit of somewhere between 2MB and 4MB. See “PROTOCOL LIMITS” in **guestfs**(3).

**tail-n**

```
tail-n nrlines path
```

If the parameter `nrlines` is a positive number, this returns the last `nrlines` lines of the file `path`.

If the parameter `nrlines` is a negative number, this returns lines from the file `path`, starting with the `-nrlines`’th line.

If the parameter `nrlines` is zero, this returns an empty list.

Because of the message protocol, there is a transfer limit of somewhere between 2MB and 4MB. See

“PROTOCOL LIMITS” in **guestfs** (3).

### **tar-in**

#### **tar-in-opts**

```
tar-in (tarfile|-) directory [compress:...] [xattrs:true|false] [selinux:true|false]
```

This command uploads and unpacks local file *tarfile* into *directory*.

The optional *compress* flag controls compression. If not given, then the input should be an uncompressed tar file. Otherwise one of the following strings may be given to select the compression type of the input file: *compress*, *gzip*, *bzip2*, *xz*, *lzop*. (Note that not all builds of *libguestfs* will support all of these compression types).

The other optional arguments are:

*xattrs*

If set to true, extended attributes are restored from the tar file.

*selinux*

If set to true, SELinux contexts are restored from the tar file.

*acls*

If set to true, POSIX ACLs are restored from the tar file.

Use *-* instead of a filename to read/write from stdin/stdout.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

### **tar-out**

#### **tar-out-opts**

```
tar-out directory (tarfile|-) [compress:...] [numericowner:true|false] [excludes:]
```

This command packs the contents of *directory* and downloads it to local file *tarfile*.

The optional *compress* flag controls compression. If not given, then the output will be an uncompressed tar file. Otherwise one of the following strings may be given to select the compression type of the output file: *compress*, *gzip*, *bzip2*, *xz*, *lzop*. (Note that not all builds of *libguestfs* will support all of these compression types).

The other optional arguments are:

*excludes*

A list of wildcards. Files are excluded if they match any of the wildcards.

*numericowner*

If set to true, the output tar file will contain UID/GID numbers instead of user/group names.

*xattrs*

If set to true, extended attributes are saved in the output tar.

*selinux*

If set to true, SELinux contexts are saved in the output tar.

*acls*

If set to true, POSIX ACLs are saved in the output tar.

Use *-* instead of a filename to read/write from stdin/stdout.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

### **tgz-in**

```
tgz-in (tarball|-) directory
```

This command uploads and unpacks local file *tarball* (a *gzip compressed* tar file) into *directory*.

Use *-* instead of a filename to read/write from stdin/stdout.

*This function is deprecated. In new code, use the “tar-in” call instead.*

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### **tgz-out**

`tgz-out directory (tarball|-)`

This command packs the contents of *directory* and downloads it to local file *tarball*.

Use `-` instead of a filename to read/write from stdin/stdout.

*This function is deprecated.* In new code, use the “tar-out” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### **touch**

`touch path`

Touch acts like the **touch**(1) command. It can be used to update the timestamps on a file, or, if the file does not exist, to create a new zero-length file.

This command only works on regular files, and will fail on other file types such as directories, symbolic links, block special etc.

### **truncate**

`truncate path`

This command truncates *path* to a zero-length file. The file must exist already.

### **truncate-size**

`truncate-size path size`

This command truncates *path* to *size* bytes. The file must exist already.

If the current file size is less than *size* then the file is extended to the required size with zero bytes. This creates a sparse file (ie. disk blocks are not allocated for the file until you write to it). To create a non-sparse file of zeroes, use “fallocate64” instead.

### **tune2fs**

`tune2fs device [force:true|false] [maxmountcount:N] [mountcount:N] [errorbehavior:continue|remount-ro|panic]`

This call allows you to adjust various filesystem parameters of an ext2/ext3/ext4 filesystem called *device*.

The optional parameters are:

#### **force**

Force **tune2fs** to complete the operation even in the face of errors. This is the same as the **tune2fs**(8) `-f` option.

#### **maxmountcount**

Set the number of mounts after which the filesystem is checked by **e2fsck**(8). If this is 0 then the number of mounts is disregarded. This is the same as the **tune2fs**(8) `-c` option.

#### **mountcount**

Set the number of times the filesystem has been mounted. This is the same as the **tune2fs**(8) `-C` option.

#### **errorbehavior**

Change the behavior of the kernel code when errors are detected. Possible values currently are: `continue`, `remount-ro`, `panic`. In practice these options don't really make any difference, particularly for write errors.

This is the same as the **tune2fs**(8) `-e` option.

#### **group**

Set the group which can use reserved filesystem blocks. This is the same as the **tune2fs**(8) `-g` option except that it can only be specified as a number.

**intervalbetweenchecks**

Adjust the maximal time between two filesystem checks (in seconds). If the option is passed as 0 then time-dependent checking is disabled.

This is the same as the **tune2fs**(8) `-i` option.

**reservedblockspercentage**

Set the percentage of the filesystem which may only be allocated by privileged processes. This is the same as the **tune2fs**(8) `-m` option.

**lastmounteddirectory**

Set the last mounted directory. This is the same as the **tune2fs**(8) `-M` option.

**reservedblockscount** Set the number of reserved filesystem blocks. This is the same as the **tune2fs**(8) `-r` option.

**user**

Set the user who can use the reserved filesystem blocks. This is the same as the **tune2fs**(8) `-u` option except that it can only be specified as a number.

To get the current values of filesystem parameters, see “**tune2fs-l**”. For precise details of how **tune2fs** works, see the **tune2fs**(8) man page.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

**tune2fs-l**

**tune2fs-l** device

This returns the contents of the ext2, ext3 or ext4 filesystem superblock on device.

It is the same as running **tune2fs -l** device. See **tune2fs**(8) manpage for more details. The list of fields returned isn’t clearly defined, and depends on both the version of **tune2fs** that **libguestfs** was built against, and the filesystem itself.

**txz-in**

**txz-in** (tarball|-) directory

This command uploads and unpacks local file **tarball** (an *xz compressed* tar file) into *directory*.

Use `-` instead of a filename to read/write from stdin/stdout.

*This function is deprecated.* In new code, use the “**tar-in**” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

This command depends on the feature **xz**. See also “feature-available”.

**txz-out**

**txz-out** directory (tarball|-)

This command packs the contents of *directory* and downloads it to local file **tarball** (as an *xz compressed* tar archive).

Use `-` instead of a filename to read/write from stdin/stdout.

*This function is deprecated.* In new code, use the “**tar-out**” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

This command depends on the feature **xz**. See also “feature-available”.

**umask**

**umask** mask

This function sets the mask used for creating new files and device nodes to **mask** & 0777.

Typical **umask** values would be 022 which creates new files with permissions like “-rw-r--r--” or “-rwxr-xr-x”, and 002 which creates new files with permissions like “-rw-rw-r--” or “-rwxrwxr-x”.



The default umask is 022. This is important because it means that directories and device nodes will be created with 0644 or 0755 mode even if you specify 0777.

See also “get-umask”, **umask** (2), “mknod”, “mkdir”.

This call returns the previous umask.

#### **umount**

#### **umount**

#### **umount-opts**

```
umount pathordevice [force:true|false] [lazyunmount:true|false]
```

This unmounts the given filesystem. The filesystem may be specified either by its mountpoint (path) or the device which contains the filesystem.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

#### **umount-all**

#### **umount-all**

```
umount-all
```

This unmounts all mounted filesystems.

Some internal mounts are not unmounted by this call.

#### **umount-local**

```
umount-local [retry:true|false]
```

If libguestfs is exporting the filesystem on a local mountpoint, then this unmounts it.

See “MOUNT LOCAL” in **guestfs** (3) for full documentation.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

#### **upload**

```
upload (filename|-) remotefilename
```

Upload local file *filename* to *remotefilename* on the filesystem.

*filename* can also be a named pipe.

See also “download”.

Use – instead of a filename to read/write from stdin/stdout.

#### **upload-offset**

```
upload-offset (filename|-) remotefilename offset
```

Upload local file *filename* to *remotefilename* on the filesystem.

*remotefilename* is overwritten starting at the byte *offset* specified. The intention is to overwrite parts of existing files or devices, although if a non-existent file is specified then it is created with a “hole” before *offset*. The size of the data written is implicit in the size of the source *filename*.

Note that there is no limit on the amount of data that can be uploaded with this call, unlike with “pwrite”, and this call always writes the full amount unless an error occurs.

See also “upload”, “pwrite”.

Use – instead of a filename to read/write from stdin/stdout.

#### **user-cancel**

```
user-cancel
```

This function cancels the current upload or download operation.

Unlike most other libguestfs calls, this function is signal safe and thread safe. You can call it from a signal handler or from another thread, without needing to do any locking.

The transfer that was in progress (if there is one) will stop shortly afterwards, and will return an error. The *errno* (see “guestfs\_last\_errno”) is set to EINTR, so you can test for this to find out if the operation was

cancelled or failed because of another error.

No cleanup is performed: for example, if a file was being uploaded then after cancellation there may be a partially uploaded file. It is the caller's responsibility to clean up if necessary.

There are two common places that you might call “user-cancel”:

In an interactive text-based program, you might call it from a `SIGINT` signal handler so that pressing `^C` cancels the current operation. (You also need to call “set-pgroup” so that child processes don't receive the `^C` signal).

In a graphical program, when the main thread is displaying a progress bar with a cancel button, wire up the cancel button to call this function.

### **utimens**

```
utimens path atsecs atnsecs mtsecs mtnsecs
```

This command sets the timestamps of a file with nanosecond precision.

`atsecs`, `atnsecs` are the last access time (`atime`) in secs and nanoseconds from the epoch.

`mtsecs`, `mtnsecs` are the last modification time (`mtime`) in secs and nanoseconds from the epoch.

If the `*nsecs` field contains the special value `-1` then the corresponding timestamp is set to the current time. (The `*secs` field is ignored in this case).

If the `*nsecs` field contains the special value `-2` then the corresponding timestamp is left unchanged. (The `*secs` field is ignored in this case).

### **utsname**

```
utsname
```

This returns the kernel version of the appliance, where this is available. This information is only useful for debugging. Nothing in the returned structure is defined by the API.

### **version**

```
version
```

Return the `libguestfs` version number that the program is linked against.

Note that because of dynamic linking this is not necessarily the version of `libguestfs` that you compiled against. You can compile the program, and then at runtime dynamically link against a completely different `libguestfs.so` library.

This call was added in version `1.0.58`. In previous versions of `libguestfs` there was no way to get the version number. From C code you can use dynamic linker functions to find out if this symbol exists (if it doesn't, then it's an earlier version).

The call returns a structure with four elements. The first three (`major`, `minor` and `release`) are numbers and correspond to the usual version triplet. The fourth element (`extra`) is a string and is normally empty, but may be used for distro-specific information.

To construct the original version string: `$major.$minor.$release$extra`

See also: “`LIBGUESTFS VERSION NUMBERS`” in `guestfs(3)`.

*Note:* Don't use this call to test for availability of features. In enterprise distributions we backport features from later versions into earlier versions, making this an unreliable way to test for features. Use “`available`” or “`feature-available`” instead.

### **vfs-label**

```
vfs-label mountable
```

This returns the label of the filesystem on `mountable`.

If the filesystem is unlabeled, this returns the empty string.

To find a filesystem from the label, use “`findfs-label`”.

**vfs-minimum-size**

```
vfs-minimum-size mountable
```

Get the minimum size of filesystem in bytes. This is the minimum possible size for filesystem shrinking.

If getting minimum size of specified filesystem is not supported, this will fail and set `errno` as `ENOTSUP`.

See also **ntfsresize** (8), **resize2fs** (8), **btrfs** (8), **xfs\_info** (8).

**vfs-type**

```
vfs-type mountable
```

This command gets the filesystem type corresponding to the filesystem on `mountable`.

For most filesystems, the result is the name of the Linux VFS module which would be used to mount this filesystem if you mounted it without specifying the filesystem type. For example a string such as `ext3` or `ntfs`.

**vfs-uuid****get-uuid**

```
vfs-uuid mountable
```

This returns the filesystem UUID of the filesystem on `mountable`.

If the filesystem does not have a UUID, this returns the empty string.

To find a filesystem from the UUID, use “`findfs-uuid`”.

**vg-activate**

```
vg-activate true|false 'volgroups ...'
```

This command activates or (if `activate` is false) deactivates all logical volumes in the listed volume groups `volgroups`.

This command is the same as running `vgchange -a y|n volgroups...`

Note that if `volgroups` is an empty list then **all** volume groups are activated or deactivated.

This command depends on the feature `lvm2`. See also “`feature-available`”.

**vg-activate-all**

```
vg-activate-all true|false
```

This command activates or (if `activate` is false) deactivates all logical volumes in all volume groups.

This command is the same as running `vgchange -a y|n`

This command depends on the feature `lvm2`. See also “`feature-available`”.

**vgchange-uuid**

```
vgchange-uuid vg
```

Generate a new random UUID for the volume group `vg`.

This command depends on the feature `lvm2`. See also “`feature-available`”.

**vgchange-uuid-all**

```
vgchange-uuid-all
```

Generate new random UUIDs for all volume groups.

This command depends on the feature `lvm2`. See also “`feature-available`”.

**vgcreate**

```
vgcreate volgroup 'physvols ...'
```

This creates an LVM volume group called `volgroup` from the non-empty list of physical volumes `physvols`.

This command depends on the feature `lvm2`. See also “`feature-available`”.

**vglvuuids**

```
vglvuuids vname
```

Given a VG called `vname`, this returns the UUIDs of all the logical volumes created in this volume group.

You can use this along with “`lvs`” and “`lvuuid`” calls to associate logical volumes and volume groups.

See also “`vgpvuuids`”.

**vgmeta**

```
vgmeta vname
```

`vname` is an LVM volume group. This command examines the volume group and returns its metadata.

Note that the metadata is an internal structure used by LVM, subject to change at any time, and is provided for information only.

This command depends on the feature `lvm2`. See also “`feature-available`”.

**vgpvuuids**

```
vgpvuuids vname
```

Given a VG called `vname`, this returns the UUIDs of all the physical volumes that this volume group resides on.

You can use this along with “`pvs`” and “`pvuuid`” calls to associate physical volumes and volume groups.

See also “`vglvuuids`”.

**vgremove**

```
vgremove vname
```

Remove an LVM volume group `vname`, (for example VG).

This also forcibly removes all logical volumes in the volume group (if any).

This command depends on the feature `lvm2`. See also “`feature-available`”.

**vgrename**

```
vgrename volgroup newvolgroup
```

Rename a volume group `volgroup` with the new name `newvolgroup`.

**vgs**

```
vgs
```

List all the volumes groups detected. This is the equivalent of the **vgs**(8) command.

This returns a list of just the volume group names that were detected (eg. `VolGroup00`).

See also “`vgs-full`”.

This command depends on the feature `lvm2`. See also “`feature-available`”.

**vgs-full**

```
vgs-full
```

List all the volumes groups detected. This is the equivalent of the **vgs**(8) command. The “full” version includes all fields.

This command depends on the feature `lvm2`. See also “`feature-available`”.

**vgscan**

```
vgscan
```

This rescans all block devices and rebuilds the list of LVM physical volumes, volume groups and logical volumes.

*This function is deprecated.* In new code, use the “`lvm-scan`” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

**vguuid**

`vguuid vname`

This command returns the UUID of the LVM VG named `vname`.

**wc-c**

`wc-c path`

This command counts the characters in a file, using the `wc -c` external command.

**wc-l**

`wc-l path`

This command counts the lines in a file, using the `wc -l` external command.

**wc-w**

`wc-w path`

This command counts the words in a file, using the `wc -w` external command.

**wipefs**

`wipefs device`

This command erases filesystem or RAID signatures from the specified `device` to make the filesystem invisible to libblkid.

This does not erase the filesystem itself nor any other data from the device.

Compare with “zero” which zeroes the first few blocks of a device.

This command depends on the feature `wipefs`. See also “feature-available”.

**write**

`write path content`

This call creates a file called `path`. The content of the file is the `stringcontent` (which can contain any 8 bit data).

See also “write-append”.

**write-append**

`write-append path content`

This call appends `content` to the end of file `path`. If `path` does not exist, then a new file is created.

See also “write”.

**write-file**

`write-file path content size`

This call creates a file called `path`. The contents of the file is the `stringcontent` (which can contain any 8 bit data), with length `size`.

As a special case, if `size` is 0 then the length is calculated using `strlen` (so in this case the content cannot contain embedded ASCII NULs).

*NB.* Owing to a bug, writing content containing ASCII NUL characters does *not* work, even if the length is specified.

Because of the message protocol, there is a transfer limit of somewhere between 2MB and 4MB. See “PROTOCOL LIMITS” in **guestfs**(3).

*This function is deprecated.* In new code, use the “write” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

**xfs-admin**

`xfs-admin device [extunwritten:true|false] [imgfile:true|false] [v2log:true|false]`

Change the parameters of the XFS filesystem on `device`.

Devices that are mounted cannot be modified. Administrators must unmount filesystems before this call can modify parameters.

Some of the parameters of a mounted filesystem can be examined and modified using the “xfs-info” and “xfs-growfs” calls.

Beginning with XFS version 5, it is no longer possible to modify the lazy-counters setting (ie. `lazycounter` parameter has no effect).

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

This command depends on the feature `xfs`. See also “feature-available”.

### **xfs-growfs**

```
xfs-growfs path [datasec:true|false] [logsec:true|false] [rtsec:true|false] [data=
```

Grow the XFS filesystem mounted at `path`.

The returned struct contains geometry information. Missing fields are returned as `-1` (for numeric fields) or empty string.

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

This command depends on the feature `xfs`. See also “feature-available”.

### **xfs-info**

```
xfs-info pathordevice
```

`pathordevice` is a mounted XFS filesystem or a device containing an XFS filesystem. This command returns the geometry of the filesystem.

The returned struct contains geometry information. Missing fields are returned as `-1` (for numeric fields) or empty string.

This command depends on the feature `xfs`. See also “feature-available”.

### **xfs-repair**

```
xfs-repair device [forcelogzero:true|false] [nomodify:true|false] [noprefetch:tr
```

Repair corrupt or damaged XFS filesystem on `device`.

The filesystem is specified using the `device` argument which should be the device name of the disk partition or volume containing the filesystem. If given the name of a block device, `xfs_repair` will attempt to find the raw device associated with the specified block device and will use the raw device instead.

Regardless, the filesystem to be repaired must be unmounted, otherwise, the resulting filesystem may be inconsistent or corrupt.

The returned status indicates whether filesystem corruption was detected (returns 1) or was not detected (returns 0).

This command has one or more optional arguments. See “OPTIONAL ARGUMENTS”.

This command depends on the feature `xfs`. See also “feature-available”.

### **yara-destroy**

```
yara-destroy
```

Destroy previously loaded Yara rules in order to free libguestfs resources.

This command depends on the feature `libyara`. See also “feature-available”.

### **yara-load**

```
yara-load (filename|-)
```

Upload a set of Yara rules from local file *filename*.

Yara rules allow to categorize files based on textual or binary patterns within their content. See “yara-scan” to see how to scan files with the loaded rules.

Rules can be in binary format, as when compiled with `yarac` command, or in source code format. In the latter case, the rules will be first compiled and then loaded.

Rules in source code format cannot include external files. In such cases, it is recommended to compile them first.

Previously loaded rules will be destroyed.

Use `-` instead of a filename to read/write from stdin/stdout.

This command depends on the feature `libyara`. See also “feature-available”.

### **yara-scan**

`yara-scan path`

Scan a file with the previously loaded Yara rules.

For each matching rule, a `yara_detection` structure is returned.

The `yara_detection` structure contains the following fields.

`yara_name`

Path of the file matching a Yara rule.

`yara_rule`

Identifier of the Yara rule which matched against the given file.

This command depends on the feature `libyara`. See also “feature-available”.

### **zegrep**

`zegrep regex path`

This calls the external `zegrep` program and returns the matching lines.

Because of the message protocol, there is a transfer limit of somewhere between 2MB and 4MB. See “PROTOCOL LIMITS” in **guestfs**(3).

*This function is deprecated.* In new code, use the “`grep`” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### **zegrepi**

`zegrepi regex path`

This calls the external `zegrep -i` program and returns the matching lines.

Because of the message protocol, there is a transfer limit of somewhere between 2MB and 4MB. See “PROTOCOL LIMITS” in **guestfs**(3).

*This function is deprecated.* In new code, use the “`grep`” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### **zero**

`zero device`

This command writes zeroes over the first few blocks of `device`.

How many blocks are zeroed isn’t specified (but it’s *not* enough to securely wipe the device). It should be sufficient to remove any partition tables, filesystem superblocks and so on.

If blocks are already zero, then this command avoids writing zeroes. This prevents the underlying device from becoming non-sparse or growing unnecessarily.

See also: “zero-device”, “scrub-device”, “is-zero-device”

### **zero-device**

`zero-device device`

This command writes zeroes over the entire device. Compare with “zero” which just zeroes the first few blocks of a device.

If blocks are already zero, then this command avoids writing zeroes. This prevents the underlying device from becoming non-sparse or growing unnecessarily.

### zero-free-space

zero-free-space *directory*

Zero the free space in the filesystem mounted on *directory*. The filesystem must be mounted read-write.

The filesystem contents are not affected, but any free space in the filesystem is freed.

Free space is not “trimmed”. You may want to call “fstrim” either as an alternative to this, or after calling this, depending on your requirements.

### zerofree

zerofree *device*

This runs the *zerofree* program on *device*. This program claims to zero unused inodes and disk blocks on an ext2/3 filesystem, thus making it possible to compress the filesystem more effectively.

You should **not** run this program if the filesystem is mounted.

It is possible that using this program can damage the filesystem or data on the filesystem.

This command depends on the feature *zerofree*. See also “feature-available”.

### zfgrep

zfgrep *pattern path*

This calls the external *zfgrep* program and returns the matching lines.

Because of the message protocol, there is a transfer limit of somewhere between 2MB and 4MB. See “PROTOCOL LIMITS” in *guestfs*(3).

*This function is deprecated.* In new code, use the “grep” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### zfgrepi

zfgrepi *pattern path*

This calls the external *zfgrep -i* program and returns the matching lines.

Because of the message protocol, there is a transfer limit of somewhere between 2MB and 4MB. See “PROTOCOL LIMITS” in *guestfs*(3).

*This function is deprecated.* In new code, use the “grep” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### zfile

zfile *meth path*

This command runs **file**(1) after first decompressing *path* using *meth*.

*meth* must be one of *gzip*, *compress* or *bzip2*.

Since 1.0.63, use “file” instead which can now process compressed files.

*This function is deprecated.* In new code, use the “file” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### zgrep

zgrep *regex path*



This calls the external **zgrep**(1) program and returns the matching lines.

Because of the message protocol, there is a transfer limit of somewhere between 2MB and 4MB. See “PROTOCOL LIMITS” in **guestfs**(3).

*This function is deprecated.* In new code, use the “grep” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

### **zgrepi**

zgrepi regex path

This calls the external **zgrep -i** program and returns the matching lines.

Because of the message protocol, there is a transfer limit of somewhere between 2MB and 4MB. See “PROTOCOL LIMITS” in **guestfs**(3).

*This function is deprecated.* In new code, use the “grep” call instead.

Deprecated functions will not be removed from the API, but the fact that they are deprecated indicates that there are problems with correct use of these functions.

## **EXIT STATUS**

guestfish returns 0 if the commands completed without error, or 1 if there was an error.

## **ENVIRONMENT VARIABLES**

### **EDITOR**

The **edit** command uses \$EDITOR as the editor. If not set, it uses **vi**.

### **GUESTFISH\_DISPLAY\_IMAGE**

The **display** command uses \$GUESTFISH\_DISPLAY\_IMAGE to display images. If not set, it uses **display**(1).

### **GUESTFISH\_INIT**

Printed when guestfish starts. See “PROMPT”.

### **GUESTFISH\_OUTPUT**

Printed before guestfish output. See “PROMPT”.

### **GUESTFISH\_PID**

Used with the **--remote** option to specify the remote guestfish process to control. See section “REMOTE CONTROL GUESTFISH OVER A SOCKET”.

### **GUESTFISH\_PS1**

Set the command prompt. See “PROMPT”.

### **GUESTFISH\_RESTORE**

Printed before guestfish exits. See “PROMPT”.

### **HEXEDITOR**

The “**hexedit**” command uses \$HEXEDITOR as the external hex editor. If not specified, the external **hexedit**(1) program is used.

### **HOME**

If compiled with GNU readline support, various files in the home directory can be used. See “FILES”.

### **LIBGUESTFS\_APPEND**

Pass additional options to the guest kernel.

### **LIBGUESTFS\_ATTACH\_METHOD**

This is the old way to set **LIBGUESTFS\_BACKEND**.

### **LIBGUESTFS\_BACKEND**

Choose the default way to create the appliance. See “**guestfs\_set\_backend**” in **guestfs**(3).

**LIBGUESTFS\_BACKEND\_SETTINGS**

A colon-separated list of backend-specific settings. See “BACKEND” in **guestfs**(3), “BACKEND SETTINGS” in **guestfs**(3).

**LIBGUESTFS\_CACHEDIR**

The location where libguestfs will cache its appliance, when using a supermin appliance. The appliance is cached and shared between all handles which have the same effective user ID.

If **LIBGUESTFS\_CACHEDIR** is not set, then **TMPDIR** is used. If **TMPDIR** is not set, then */var/tmp* is used.

See also “**LIBGUESTFS\_TMPDIR**”, “**set-cachedir**”.

**LIBGUESTFS\_DEBUG**

Set **LIBGUESTFS\_DEBUG=1** to enable verbose messages. This has the same effect as using the **-v** option.

**LIBGUESTFS\_HV**

Set the default hypervisor (usually qemu) binary that libguestfs uses. If not set, then the qemu which was found at compile time by the configure script is used.

**LIBGUESTFS\_MEMSIZE**

Set the memory allocated to the qemu process, in megabytes. For example:

```
LIBGUESTFS_MEMSIZE=700
```

**LIBGUESTFS\_PATH**

Set the path that guestfish uses to search for kernel and initrd.img. See the discussion of paths in **guestfs**(3).

**LIBGUESTFS\_QEMU**

This is the old way to set **LIBGUESTFS\_HV**.

**LIBGUESTFS\_TMPDIR**

The location where libguestfs will store temporary files used by each handle.

If **LIBGUESTFS\_TMPDIR** is not set, then **TMPDIR** is used. If **TMPDIR** is not set, then */tmp* is used.

See also “**LIBGUESTFS\_CACHEDIR**”, “**set-tmpdir**”.

**LIBGUESTFS\_TRACE**

Set **LIBGUESTFS\_TRACE=1** to enable command traces.

**PAGER**

The **more** command uses **\$PAGER** as the pager. If not set, it uses **more**.

**PATH**

Libguestfs and guestfish may run some external programs, and rely on **\$PATH** being set to a reasonable value. If using the libvirt backend, libvirt will not work at all unless **\$PATH** contains the path of qemu/KVM.

**SUPERMIN\_KERNEL****SUPERMIN\_KERNEL\_VERSION****SUPERMIN\_MODULES**

These three environment variables allow the kernel that libguestfs uses in the appliance to be selected.

If **\$SUPERMIN\_KERNEL** is not set, then the most recent host kernel is chosen. For more information about kernel selection, see **supermin**(1).

**TMPDIR**

See “**LIBGUESTFS\_CACHEDIR**”, “**LIBGUESTFS\_TMPDIR**”.

**XDG\_RUNTIME\_DIR**

This directory represents a user-specific directory for storing non-essential runtime files.

If it is set, then is used to store temporary sockets. Otherwise */tmp* is used.

See also “get-sockdir”, <http://www.freedesktop.org/wiki/Specifications/basedir-spec/>.

## FILES

`$XDG_CONFIG_HOME/libguestfs/libguestfs-tools.conf`

`$HOME/.libguestfs-tools.rc`

`$XDG_CONFIG_DIRS/libguestfs/libguestfs-tools.conf`

`/etc/libguestfs-tools.conf`

This configuration file controls the default read-only or read-write mode (`--ro` or `--rw`).

See **libguestfs-tools.conf** (5).

`$HOME/.guestfish`

If compiled with GNU readline support, then the command history is saved in this file.

`$HOME/.inputrc`

`/etc/inputrc`

If compiled with GNU readline support, then these files can be used to configure readline. For further information, please see “INITIALIZATION FILE” in **readline** (3).

To write rules which only apply to guestfish, use:

```
$if guestfish
...
$endif
```

Variables that you can set in inputrc that change the behaviour of guestfish in useful ways include:

completion-ignore-case (default: on)

By default, guestfish will ignore case when tab-completing paths on the disk. Use:

```
set completion-ignore-case off
```

to make guestfish case sensitive.

`test1.img`

`test2.img` (etc)

When using the `-N` or `--new` option, the prepared disk or filesystem will be created in the file *test1.img* in the current directory. The second use of `-N` will use *test2.img* and so on. Any existing file with the same name will be overwritten. You can use a different filename by using the `filename=` prefix.

## SEE ALSO

**guestfs** (3), <http://libguestfs.org/>, **virt-alignment-scan** (1), **virt-builder** (1), **virt-builder-repository** (1), **virt-cat** (1), **virt-copy-in** (1), **virt-copy-out** (1), **virt-customize** (1), **virt-df** (1), **virt-diff** (1), **virt-edit** (1), **virt-filesystems** (1), **virt-inspector** (1), **virt-list-filesystems** (1), **virt-list-partitions** (1), **virt-log** (1), **virt-ls** (1), **virt-make-fs** (1), **virt-p2v** (1), **virt-rescue** (1), **virt-resize** (1), **virt-sparsify** (1), **virt-sysprep** (1), **virt-tail** (1), **virt-tar** (1), **virt-tar-in** (1), **virt-tar-out** (1), **virt-v2v** (1), **virt-win-reg** (1), **libguestfs-tools.conf** (5), **display** (1), **hexedit** (1), **supermin** (1).

## AUTHORS

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

## COPYRIGHT

Copyright (C) 2009–2020 Red Hat Inc.

## LICENSE

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

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

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

## 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](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.