QEMU Emulator
User Documentation

Table of Contents

1 Introduction

1.1 Features

QEMU is a FAST! processor emulator using dynamic translation to achieve good emulation speed.

QEMU has two operating modes:

- Full system emulation. In this mode, QEMU emulates a full system (for example a PC), including one or several processors and various peripherals. It can be used to launch different Operating Systems without rebooting the PC or to debug system code.
- User mode emulation. In this mode, QEMU can launch processes compiled for one CPU on another CPU. It can be used to launch the Wine Windows API emulator (`http://www.winehq.org`) or to ease cross-compilation and cross-debugging.

QEMU has the following features:

- QEMU can run without a host kernel driver and yet gives acceptable performance. It uses dynamic translation to native code for reasonable speed, with support for self-modifying code and precise exceptions.
- It is portable to several operating systems (GNU/Linux, *BSD, Mac OS X, Windows) and architectures.
- It performs accurate software emulation of the FPU.

QEMU user mode emulation has the following features:

- Generic Linux system call converter, including most ioctls.
- clone() emulation using native CPU clone() to use Linux scheduler for threads.
- Accurate signal handling by remapping host signals to target signals.

QEMU full system emulation has the following features:

- QEMU uses a full software MMU for maximum portability.
- QEMU can optionally use an in-kernel accelerator, like kvm. The accelerators execute most of the guest code natively, while continuing to emulate the rest of the machine.
- Various hardware devices can be emulated and in some cases, host devices (e.g. serial and parallel ports, USB, drives) can be used transparently by the guest Operating System. Host device passthrough can be used for talking to external physical peripherals (e.g. a webcam, modem or tape drive).
- Symmetric multiprocessing (SMP) support. Currently, an in-kernel accelerator is required to use more than one host CPU for emulation.

2 QEMU PC System emulator

2.1 Introduction

The QEMU PC System emulator simulates the following peripherals:

- i440FX host PCI bridge and PIIX3 PCI to ISA bridge
- Cirrus CLGD 5446 PCI VGA card or dummy VGA card with Bochs VESA extensions (hardware level, including all non standard modes).
- PS/2 mouse and keyboard
- 2 PCI IDE interfaces with hard disk and CD-ROM support
- Floppy disk
- PCI and ISA network adapters
- Serial ports
- IPMI BMC, either and internal or external one
- Creative SoundBlaster 16 sound card
- ENSONIQ AudioPCI ES1370 sound card
- Intel 82801AA AC97 Audio compatible sound card
- Intel HD Audio Controller and HDA codec
- Adlib (OPL2) - Yamaha YM3812 compatible chip
- Gravis Ultrasound GF1 sound card
- CS4231A compatible sound card
- PCI UHCI USB controller and a virtual USB hub.

SMP is supported with up to 255 CPUs.

QEMU uses the PC BIOS from the Seabios project and the Plex86/Bochs LGPL VGA BIOS.

QEMU uses YM3812 emulation by Tatsuyuki Satoh.

QEMU uses GUS emulation (GUSEMU32 `http://www.deinmeister.de/gusemu/`) by Tibor "TS" Schütz.

Note that, by default, GUS shares IRQ(7) with parallel ports and so QEMU must be told to not have parallel ports to have working GUS.

```
qemu-system-i386 dos.img -soundhw gus -parallel none
```

Alternatively:

```
qemu-system-i386 dos.img -device gus,irq=5
```

Or some other unclaimed IRQ.

CS4231A is the chip used in Windows Sound System and GUSMAX products

2.2 Quick Start

Download and uncompress the linux image (`linux.img`) and type:

```
qemu-system-i386 linux.img
```

Linux should boot and give you a prompt.

2.3 Invocation

`qemu-system-i386 [`*options*`] [`*disk_image*`]`

disk_image is a raw hard disk image for IDE hard disk 0. Some targets do not need a disk image.

Standard options:

`-h` Display help and exit

`-version` Display version information and exit

`-machine [type=]`*name*`[,prop=`*value*`[,...]]`
Select the emulated machine by *name*. Use `-machine help` to list available machines. Supported machine properties are:

`accel=`*accels1*`[:`*accels2*`[:...]]`
This is used to enable an accelerator. Depending on the target architecture, kvm, xen, or tcg can be available. By default, tcg is used. If there is more than one accelerator specified, the next one is used if the previous one fails to initialize.

`kernel_irqchip=on|off`
Controls in-kernel irqchip support for the chosen accelerator when available.

`gfx_passthru=on|off`
Enables IGD GFX passthrough support for the chosen machine when available.

`vmport=on|off|auto`
Enables emulation of VMWare IO port, for vmmouse etc. auto says to select the value based on accel. For accel=xen the default is off otherwise the default is on.

`kvm_shadow_mem=size`
Defines the size of the KVM shadow MMU.

`dump-guest-core=on|off`
Include guest memory in a core dump. The default is on.

`mem-merge=on|off`
Enables or disables memory merge support. This feature, when supported by the host, de-duplicates identical memory pages among VMs instances (enabled by default).

`aes-key-wrap=on|off`
Enables or disables AES key wrapping support on s390-ccw hosts. This feature controls whether AES wrapping keys will be created to allow execution of AES cryptographic functions. The default is on.

`dea-key-wrap=on|off`
Enables or disables DEA key wrapping support on s390-ccw hosts. This feature controls whether DEA wrapping keys will be created

to allow execution of DEA cryptographic functions. The default is on.

`nvdimm=on|off`
Enables or disables NVDIMM support. The default is off.

`-cpu` *`model`*
Select CPU model (`-cpu help` for list and additional feature selection)

`-smp`
`[cpus=]`*`n`*`[,cores=`*`cores`*`][,threads=`*`threads`*`][,sockets=`*`sockets`*`][,maxcpus=`*`maxcpus`*`]`
Simulate an SMP system with *n* CPUs. On the PC target, up to 255 CPUs are supported. On Sparc32 target, Linux limits the number of usable CPUs to 4. For the PC target, the number of *cores* per socket, the number of *threads* per cores and the total number of *sockets* can be specified. Missing values will be computed. If any on the three values is given, the total number of CPUs *n* can be omitted. *maxcpus* specifies the maximum number of hotpluggable CPUs.

`-numa node[,mem=`*`size`*`][,cpus=`*`cpu[-cpu]`*`][,nodeid=`*`node`*`]`
`-numa node[,memdev=`*`id`*`][,cpus=`*`cpu[-cpu]`*`][,nodeid=`*`node`*`]`
Simulate a multi node NUMA system. If '`mem`', '`memdev`' and '`cpus`' are omitted, resources are split equally. Also, note that the `-numa` option doesn't allocate any of the specified resources. That is, it just assigns existing resources to NUMA nodes. This means that one still has to use the `-m`, `-smp` options to allocate RAM and VCPUs respectively, and possibly `-object` to specify the memory backend for the '`memdev`' suboption.

'`mem`' and '`memdev`' are mutually exclusive. Furthermore, if one node uses '`memdev`', all of them have to use it.

`-add-fd fd=`*`fd`*`,set=`*`set`*`[,opaque=`*`opaque`*`]`
Add a file descriptor to an fd set. Valid options are:

`fd=`*`fd`*
This option defines the file descriptor of which a duplicate is added to fd set. The file descriptor cannot be stdin, stdout, or stderr.

`set=`*`set`*
This option defines the ID of the fd set to add the file descriptor to.

`opaque=`*`opaque`*
This option defines a free-form string that can be used to describe *fd*.

You can open an image using pre-opened file descriptors from an fd set:

```
qemu-system-i386
-add-fd fd=3,set=2,opaque="rdwr:/path/to/file"
-add-fd fd=4,set=2,opaque="rdonly:/path/to/file"
-drive file=/dev/fdset/2,index=0,media=disk
```

`-set` *`group.id.arg=value`*
Set parameter *arg* for item *id* of type *group*

`-global` *driver.prop=value*
`-global driver=`*driver*`,property=`*property*`,value=`*value*

Set default value of *driver*'s property *prop* to *value*, e.g.:

```
qemu-system-i386 -global ide-drive.physical_block_size=4096 -drive file=file,if=i
```

In particular, you can use this to set driver properties for devices which are created automatically by the machine model. To create a device which is not created automatically and set properties on it, use `-device`.

-global *driver.prop=value* is shorthand for -global driver=*driver*,property=*prop*,value=*value*. The longhand syntax works even when *driver* contains a dot.

`-boot [order=`*drives*`][,once=`*drives*`][,menu=on|off][,splash=`*sp_name*`][,splash-time=`*sp_time*`][,reboot-timeout=`*rb_timeout*`][,strict=on|off]`

Specify boot order *drives* as a string of drive letters. Valid drive letters depend on the target architecture. The x86 PC uses: a, b (floppy 1 and 2), c (first hard disk), d (first CD-ROM), n-p (Etherboot from network adapter 1-4), hard disk boot is the default. To apply a particular boot order only on the first startup, specify it via `once`.

Interactive boot menus/prompts can be enabled via `menu=on` as far as firmware/BIOS supports them. The default is non-interactive boot.

A splash picture could be passed to bios, enabling user to show it as logo, when option splash=*sp_name* is given and menu=on, If firmware/BIOS supports them. Currently Seabios for X86 system support it. limitation: The splash file could be a jpeg file or a BMP file in 24 BPP format(true color). The resolution should be supported by the SVGA mode, so the recommended is 320x240, 640x480, 800x640.

A timeout could be passed to bios, guest will pause for *rb_timeout* ms when boot failed, then reboot. If *rb_timeout* is '-1', guest will not reboot, qemu passes '-1' to bios by default. Currently Seabios for X86 system support it.

Do strict boot via `strict=on` as far as firmware/BIOS supports it. This only effects when boot priority is changed by bootindex options. The default is non-strict boot.

```
# try to boot from network first, then from hard disk
qemu-system-i386 -boot order=nc
# boot from CD-ROM first, switch back to default order after reboot
qemu-system-i386 -boot once=d
# boot with a splash picture for 5 seconds.
qemu-system-i386 -boot menu=on,splash=/root/boot.bmp,splash-time=5000
```

Note: The legacy format '-boot *drives*' is still supported but its use is discouraged as it may be removed from future versions.

`-m [size=]`*megs*`[,slots=n,maxmem=size]`

Sets guest startup RAM size to *megs* megabytes. Default is 128 MiB. Optionally, a suffix of "M" or "G" can be used to signify a value in megabytes or gigabytes respectively. Optional pair *slots*, *maxmem* could be used to set amount of hotpluggable memory slots and maximum amount of memory. Note that *maxmem* must be aligned to the page size.

For example, the following command-line sets the guest startup RAM size to 1GB, creates 3 slots to hotplug additional memory and sets the maximum memory the guest can reach to 4GB:

```
qemu-system-x86_64 -m 1G,slots=3,maxmem=4G
```

If *slots* and *maxmem* are not specified, memory hotplug won't be enabled and the guest startup RAM will never increase.

`-mem-path` *path*
Allocate guest RAM from a temporarily created file in *path*.

`-mem-prealloc`
Preallocate memory when using -mem-path.

`-k` *language*
Use keyboard layout *language* (for example `fr` for French). This option is only needed where it is not easy to get raw PC keycodes (e.g. on Macs, with some X11 servers or with a VNC or curses display). You don't normally need to use it on PC/Linux or PC/Windows hosts.

The available layouts are:

```
ar  de-ch  es  fo     fr-ca  hu  ja  mk     no  pt-br  sv
da  en-gb  et  fr     fr-ch  is  lt  nl     pl  ru     th
de  en-us  fi  fr-be  hr     it  lv  nl-be  pt  sl     tr
```

The default is `en-us`.

`-audio-help`
Will show the audio subsystem help: list of drivers, tunable parameters.

`-soundhw` *card1*`[,`*card2*`,...] or -soundhw all`
Enable audio and selected sound hardware. Use 'help' to print all available sound hardware.

```
qemu-system-i386 -soundhw sb16,adlib disk.img
qemu-system-i386 -soundhw es1370 disk.img
qemu-system-i386 -soundhw ac97 disk.img
qemu-system-i386 -soundhw hda disk.img
qemu-system-i386 -soundhw all disk.img
qemu-system-i386 -soundhw help
```

Note that Linux's i810_audio OSS kernel (for AC97) module might require manually specifying clocking.

```
modprobe i810_audio clocking=48000
```

`-balloon none`
Disable balloon device.

`-balloon virtio[,addr=`*addr*`]`
Enable virtio balloon device (default), optionally with PCI address *addr*.

`-device` *driver*`[,`*prop*`[=`*value*`][,...]]`
Add device *driver*. *prop*=*value* sets driver properties. Valid properties depend on the driver. To get help on possible drivers and properties, use `-device help` and `-device` *driver*`,help`.

Some drivers are:

`-device ipmi-bmc-sim,id=`*id*`[,slave_addr=`*val*`]`

Add an IPMI BMC. This is a simulation of a hardware management interface processor that normally sits on a system. It provides a watchdog and the ability to reset and power control the system. You need to connect this to an IPMI interface to make it useful

The IPMI slave address to use for the BMC. The default is 0x20. This address is the BMC's address on the I2C network of management controllers. If you don't know what this means, it is safe to ignore it.

`-device ipmi-bmc-extern,id=`*id*`,chardev=`*id*`[,slave_addr=`*val*`]`

Add a connection to an external IPMI BMC simulator. Instead of locally emulating the BMC like the above item, instead connect to an external entity that provides the IPMI services.

A connection is made to an external BMC simulator. If you do this, it is strongly recommended that you use the "reconnect=" chardev option to reconnect to the simulator if the connection is lost. Note that if this is not used carefully, it can be a security issue, as the interface has the ability to send resets, NMIs, and power off the VM. It's best if QEMU makes a connection to an external simulator running on a secure port on localhost, so neither the simulator nor QEMU is exposed to any outside network.

See the "lanserv/README.vm" file in the OpenIPMI library for more details on the external interface.

`-device isa-ipmi-kcs,bmc=`*id*`[,ioport=`*val*`][,irq=`*val*`]`

Add a KCS IPMI interafce on the ISA bus. This also adds a corresponding ACPI and SMBIOS entries, if appropriate.

`bmc=`*id* The BMC to connect to, one of ipmi-bmc-sim or ipmi-bmc-extern above.

`ioport=`*val*

Define the I/O address of the interface. The default is 0xca0 for KCS.

`irq=`*val* Define the interrupt to use. The default is 5. To disable interrupts, set this to 0.

`-device isa-ipmi-bt,bmc=`*id*`[,ioport=`*val*`][,irq=`*val*`]`

Like the KCS interface, but defines a BT interface. The default port is 0xe4 and the default interrupt is 5.

`-name` *name*

Sets the *name* of the guest. This name will be displayed in the SDL window caption. The *name* will also be used for the VNC server. Also optionally set the top visible process name in Linux. Naming of individual threads can also be enabled on Linux to aid debugging.

`-uuid` *uuid*

Set system UUID.

Block device options:

`-fda` *file*
`-fdb` *file* Use *file* as floppy disk 0/1 image (see Section 2.7 [disk_images], page 64).

`-hda` *file*
`-hdb` *file*
`-hdc` *file*
`-hdd` *file* Use *file* as hard disk 0, 1, 2 or 3 image (see Section 2.7 [disk_images], page 64).

`-cdrom` *file*
Use *file* as CD-ROM image (you cannot use `-hdc` and `-cdrom` at the same time). You can use the host CD-ROM by using `/dev/cdrom` as filename (see Section 2.7.8 [host_drives], page 81).

`-drive` *option*`[,`*option*`[,`*option*`[,...]]]`
Define a new drive. Valid options are:

`file=`*file*
This option defines which disk image (see Section 2.7 [disk_images], page 64) to use with this drive. If the filename contains comma, you must double it (for instance, "file=my,,file" to use file "my,file").

Special files such as iSCSI devices can be specified using protocol specific URLs. See the section for "Device URL Syntax" for more information.

`if=`*interface*
This option defines on which type on interface the drive is connected. Available types are: ide, scsi, sd, mtd, floppy, pflash, virtio.

`bus=`*bus*`,unit=`*unit*
These options define where is connected the drive by defining the bus number and the unit id.

`index=`*index*
This option defines where is connected the drive by using an index in the list of available connectors of a given interface type.

`media=`*media*
This option defines the type of the media: disk or cdrom.

`cyls=`*c*`,heads=`*h*`,secs=`*s*`[,trans=`*t*`]`
These options have the same definition as they have in `-hdachs`.

`snapshot=`*snapshot*
snapshot is "on" or "off" and controls snapshot mode for the given drive (see `-snapshot`).

`cache=`*cache*
cache is "none", "writeback", "unsafe", "directsync" or "writethrough" and controls how the host cache is used to access block data.

`aio=`*aio* *aio* is "threads", or "native" and selects between pthread based disk I/O and native Linux AIO.

`discard=`*discard*
: *discard* is one of "ignore" (or "off") or "unmap" (or "on") and controls whether *discard* (also known as *trim* or *unmap*) requests are ignored or passed to the filesystem. Some machine types may not support discard requests.

`format=`*format*
: Specify which disk *format* will be used rather than detecting the format. Can be used to specify format=raw to avoid interpreting an untrusted format header.

`serial=`*serial*
: This option specifies the serial number to assign to the device.

`addr=`*addr*
: Specify the controller's PCI address (if=virtio only).

`werror=`*action*`,rerror=`*action*
: Specify which *action* to take on write and read errors. Valid actions are: "ignore" (ignore the error and try to continue), "stop" (pause QEMU), "report" (report the error to the guest), "enospc" (pause QEMU only if the host disk is full; report the error to the guest otherwise). The default setting is `werror=enospc` and `rerror=report`.

`readonly`
: Open drive `file` as read-only. Guest write attempts will fail.

`copy-on-read=`*copy-on-read*
: *copy-on-read* is "on" or "off" and enables whether to copy read backing file sectors into the image file.

`detect-zeroes=`*detect-zeroes*
: *detect-zeroes* is "off", "on" or "unmap" and enables the automatic conversion of plain zero writes by the OS to driver specific optimized zero write commands. You may even choose "unmap" if *discard* is set to "unmap" to allow a zero write to be converted to an UNMAP operation.

By default, the `cache=writeback` mode is used. It will report data writes as completed as soon as the data is present in the host page cache. This is safe as long as your guest OS makes sure to correctly flush disk caches where needed. If your guest OS does not handle volatile disk write caches correctly and your host crashes or loses power, then the guest may experience data corruption.

For such guests, you should consider using `cache=writethrough`. This means that the host page cache will be used to read and write data, but write notification will be sent to the guest only after QEMU has made sure to flush each write to the disk. Be aware that this has a major impact on performance.

The host page cache can be avoided entirely with `cache=none`. This will attempt to do disk IO directly to the guest's memory. QEMU may still perform an internal copy of the data. Note that this is considered a writeback mode and the guest OS must handle the disk write cache correctly in order to avoid data corruption on host crashes.

The host page cache can be avoided while only sending write notifications to the guest when the data has been flushed to the disk using `cache=directsync`.

In case you don't care about data integrity over host failures, use `cache=unsafe`. This option tells QEMU that it never needs to write any data to the disk but can instead keep things in cache. If anything goes wrong, like your host losing power, the disk storage getting disconnected accidentally, etc. your image will most probably be rendered unusable. When using the `-snapshot` option, unsafe caching is always used.

Copy-on-read avoids accessing the same backing file sectors repeatedly and is useful when the backing file is over a slow network. By default copy-on-read is off.

Instead of `-cdrom` you can use:

```
qemu-system-i386 -drive file=file,index=2,media=cdrom
```

Instead of `-hda`, `-hdb`, `-hdc`, `-hdd`, you can use:

```
qemu-system-i386 -drive file=file,index=0,media=disk
qemu-system-i386 -drive file=file,index=1,media=disk
qemu-system-i386 -drive file=file,index=2,media=disk
qemu-system-i386 -drive file=file,index=3,media=disk
```

You can open an image using pre-opened file descriptors from an fd set:

```
qemu-system-i386
-add-fd fd=3,set=2,opaque="rdwr:/path/to/file"
-add-fd fd=4,set=2,opaque="rdonly:/path/to/file"
-drive file=/dev/fdset/2,index=0,media=disk
```

You can connect a CDROM to the slave of ide0:

```
qemu-system-i386 -drive file=file,if=ide,index=1,media=cdrom
```

If you don't specify the "file=" argument, you define an empty drive:

```
qemu-system-i386 -drive if=ide,index=1,media=cdrom
```

You can connect a SCSI disk with unit ID 6 on the bus #0:

```
qemu-system-i386 -drive file=file,if=scsi,bus=0,unit=6
```

Instead of `-fda`, `-fdb`, you can use:

```
qemu-system-i386 -drive file=file,index=0,if=floppy
qemu-system-i386 -drive file=file,index=1,if=floppy
```

By default, *interface* is "ide" and *index* is automatically incremented:

```
qemu-system-i386 -drive file=a -drive file=b"
```

is interpreted like:

```
qemu-system-i386 -hda a -hdb b
```

`-mtdblock` *file*
: Use *file* as on-board Flash memory image.

`-sd` *file* Use *file* as SecureDigital card image.

`-pflash` *file*
: Use *file* as a parallel flash image.

`-snapshot`
: Write to temporary files instead of disk image files. In this case, the raw disk image you use is not written back. You can however force the write back by pressing `C-a s` (see Section 2.7 [disk_images], page 64).

`-hdachs` *c*`,`*h*`,`*s*`,[,`*t*`]`
: Force hard disk 0 physical geometry (1 <= *c* <= 16383, 1 <= *h* <= 16, 1 <= *s* <= 63) and optionally force the BIOS translation mode (*t*=none, lba or auto). Usually QEMU can guess all those parameters. This option is useful for old MS-DOS disk images.

`-fsdev` *fsdriver*`,id=`*id*`,path=`*path*`,[security_model=`*security_model*`][,writeout=`*writeout*`][,readonly][,socket=`*socket*`|sock_fd=`*sock_fd*`]`
: Define a new file system device. Valid options are:

  fsdriver
  : This option specifies the fs driver backend to use. Currently "local", "handle" and "proxy" file system drivers are supported.

  `id=`*id*
  : Specifies identifier for this device

  `path=`*path*
  : Specifies the export path for the file system device. Files under this path will be available to the 9p client on the guest.

  `security_model=`*security_model*
  : Specifies the security model to be used for this export path. Supported security models are "passthrough", "mapped-xattr", "mapped-file" and "none". In "passthrough" security model, files are stored using the same credentials as they are created on the guest. This requires QEMU to run as root. In "mapped-xattr" security model, some of the file attributes like uid, gid, mode bits and link target are stored as file attributes. For "mapped-file" these attributes are stored in the hidden .virtfs_metadata directory. Directories exported by this security model cannot interact with other unix tools. "none" security model is same as passthrough except the sever won't report failures if it fails to set file attributes like ownership. Security model is mandatory only for local fsdriver. Other fsdrivers (like handle, proxy) don't take security model as a parameter.

  `writeout=`*writeout*
  : This is an optional argument. The only supported value is "immediate". This means that host page cache will be used to read and write data but write notification will be sent to the guest only when the data has been reported as written by the storage subsystem.

  `readonly`
  : Enables exporting 9p share as a readonly mount for guests. By default read-write access is given.

  `socket=`*socket*
  : Enables proxy filesystem driver to use passed socket file for communicating with virtfs-proxy-helper

`sock_fd=`*`sock_fd`*
Enables proxy filesystem driver to use passed socket descriptor for communicating with virtfs-proxy-helper. Usually a helper like libvirt will create socketpair and pass one of the fds as sock_fd

-fsdev option is used along with -device driver "virtio-9p-pci".

`-device virtio-9p-pci,fsdev=`*`id`*`,mount_tag=`*`mount_tag`*
Options for virtio-9p-pci driver are:

`fsdev=`*`id`* Specifies the id value specified along with -fsdev option

`mount_tag=`*`mount_tag`*
Specifies the tag name to be used by the guest to mount this export point

`-virtfs` *`fsdriver`*`[,path=`*`path`*`],mount_tag=`*`mount_tag`*`[,security_model=`*`security_model`*`][,writeout=`*`writeout`*`][,readonly][,socket=`*`socket`*`|sock_fd=`*`sock_fd`*`]`
The general form of a Virtual File system pass-through options are:

`fsdriver` This option specifies the fs driver backend to use. Currently "local", "handle" and "proxy" file system drivers are supported.

`id=`*`id`* Specifies identifier for this device

`path=`*`path`*
Specifies the export path for the file system device. Files under this path will be available to the 9p client on the guest.

`security_model=`*`security_model`*
Specifies the security model to be used for this export path. Supported security models are "passthrough", "mapped-xattr", "mapped-file" and "none". In "passthrough" security model, files are stored using the same credentials as they are created on the guest. This requires QEMU to run as root. In "mapped-xattr" security model, some of the file attributes like uid, gid, mode bits and link target are stored as file attributes. For "mapped-file" these attributes are stored in the hidden .virtfs_metadata directory. Directories exported by this security model cannot interact with other unix tools. "none" security model is same as passthrough except the sever won't report failures if it fails to set file attributes like ownership. Security model is mandatory only for local fsdriver. Other fsdrivers (like handle, proxy) don't take security model as a parameter.

`writeout=`*`writeout`*
This is an optional argument. The only supported value is "immediate". This means that host page cache will be used to read and write data but write notification will be sent to the guest only when the data has been reported as written by the storage subsystem.

`readonly` Enables exporting 9p share as a readonly mount for guests. By default read-write access is given.

`socket=`*`socket`*
: Enables proxy filesystem driver to use passed socket file for communicating with virtfs-proxy-helper. Usually a helper like libvirt will create socketpair and pass one of the fds as sock_fd

`sock_fd`
: Enables proxy filesystem driver to use passed 'sock_fd' as the socket descriptor for interfacing with virtfs-proxy-helper

`-virtfs_synth`
: Create synthetic file system image

USB options:

`-usb`
: Enable the USB driver (will be the default soon)

`-usbdevice` *`devname`*
: Add the USB device *devname*. See Section 2.11.1 [usb_devices], page 90.

`mouse`
: Virtual Mouse. This will override the PS/2 mouse emulation when activated.

`tablet`
: Pointer device that uses absolute coordinates (like a touchscreen). This means QEMU is able to report the mouse position without having to grab the mouse. Also overrides the PS/2 mouse emulation when activated.

`disk:[format=`*`format`*`]:`*`file`*
: Mass storage device based on file. The optional *format* argument will be used rather than detecting the format. Can be used to specify `format=raw` to avoid interpreting an untrusted format header.

`host:`*`bus.addr`*
: Pass through the host device identified by *bus.addr* (Linux only).

`host:`*`vendor_id`*`:`*`product_id`*
: Pass through the host device identified by *vendor_id*:*product_id* (Linux only).

`serial:[vendorid=`*`vendor_id`*`][,productid=`*`product_id`*`]:`*`dev`*
: Serial converter to host character device *dev*, see `-serial` for the available devices.

`braille`
: Braille device. This will use BrlAPI to display the braille output on a real or fake device.

`net:`*`options`*
: Network adapter that supports CDC ethernet and RNDIS protocols.

Display options:

`-display` *`type`*
: Select type of display to use. This option is a replacement for the old style -sdl/-curses/... options. Valid values for *type* are

`sdl`
: Display video output via SDL (usually in a separate graphics window; see the SDL documentation for other possibilities).

`curses` Display video output via curses. For graphics device models which support a text mode, QEMU can display this output using a curses/ncurses interface. Nothing is displayed when the graphics device is in graphical mode or if the graphics device does not support a text mode. Generally only the VGA device models support text mode.

`none` Do not display video output. The guest will still see an emulated graphics card, but its output will not be displayed to the QEMU user. This option differs from the -nographic option in that it only affects what is done with video output; -nographic also changes the destination of the serial and parallel port data.

`gtk` Display video output in a GTK window. This interface provides drop-down menus and other UI elements to configure and control the VM during runtime.

`vnc` Start a VNC server on display <arg>

`-nographic`

Normally, if QEMU is compiled with graphical window support, it displays output such as guest graphics, guest console, and the QEMU monitor in a window. With this option, you can totally disable graphical output so that QEMU is a simple command line application. The emulated serial port is redirected on the console and muxed with the monitor (unless redirected elsewhere explicitly). Therefore, you can still use QEMU to debug a Linux kernel with a serial console. Use `C-a h` for help on switching between the console and monitor.

`-curses` Normally, if QEMU is compiled with graphical window support, it displays output such as guest graphics, guest console, and the QEMU monitor in a window. With this option, QEMU can display the VGA output when in text mode using a curses/ncurses interface. Nothing is displayed in graphical mode.

`-no-frame`

Do not use decorations for SDL windows and start them using the whole available screen space. This makes the using QEMU in a dedicated desktop workspace more convenient.

`-alt-grab`

Use Ctrl-Alt-Shift to grab mouse (instead of Ctrl-Alt). Note that this also affects the special keys (for fullscreen, monitor-mode switching, etc).

`-ctrl-grab`

Use Right-Ctrl to grab mouse (instead of Ctrl-Alt). Note that this also affects the special keys (for fullscreen, monitor-mode switching, etc).

`-no-quit` Disable SDL window close capability.

`-sdl` Enable SDL.

`-spice` *option*`[,`*option*`[,...]]`

Enable the spice remote desktop protocol. Valid options are

`port=<nr>`

Set the TCP port spice is listening on for plaintext channels.

`addr=<addr>`
: Set the IP address spice is listening on. Default is any address.

`ipv4`
`ipv6`
`unix`
: Force using the specified IP version.

`password=<secret>`
: Set the password you need to authenticate.

`sasl`
: Require that the client use SASL to authenticate with the spice. The exact choice of authentication method used is controlled from the system / user's SASL configuration file for the 'qemu' service. This is typically found in /etc/sasl2/qemu.conf. If running QEMU as an unprivileged user, an environment variable SASL_CONF_PATH can be used to make it search alternate locations for the service config. While some SASL auth methods can also provide data encryption (eg GSSAPI), it is recommended that SASL always be combined with the 'tls' and 'x509' settings to enable use of SSL and server certificates. This ensures a data encryption preventing compromise of authentication credentials.

`disable-ticketing`
: Allow client connects without authentication.

`disable-copy-paste`
: Disable copy paste between the client and the guest.

`disable-agent-file-xfer`
: Disable spice-vdagent based file-xfer between the client and the guest.

`tls-port=<nr>`
: Set the TCP port spice is listening on for encrypted channels.

`x509-dir=<dir>`
: Set the x509 file directory. Expects same filenames as -vnc $display,x509=$dir

`x509-key-file=<file>`
`x509-key-password=<file>`
`x509-cert-file=<file>`
`x509-cacert-file=<file>`
`x509-dh-key-file=<file>`
: The x509 file names can also be configured individually.

`tls-ciphers=<list>`
: Specify which ciphers to use.

`tls-channel=[main|display|cursor|inputs|record|playback]`
`plaintext-channel=[main|display|cursor|inputs|record|playback]`
: Force specific channel to be used with or without TLS encryption. The options can be specified multiple times to configure multiple

channels. The special name "default" can be used to set the default mode. For channels which are not explicitly forced into one mode the spice client is allowed to pick tls/plaintext as he pleases.

`image-compression=[auto_glz|auto_lz|quic|glz|lz|off]`
Configure image compression (lossless). Default is auto_glz.

`jpeg-wan-compression=[auto|never|always]`
`zlib-glz-wan-compression=[auto|never|always]`
Configure wan image compression (lossy for slow links). Default is auto.

`streaming-video=[off|all|filter]`
Configure video stream detection. Default is off.

`agent-mouse=[on|off]`
Enable/disable passing mouse events via vdagent. Default is on.

`playback-compression=[on|off]`
Enable/disable audio stream compression (using celt 0.5.1). Default is on.

`seamless-migration=[on|off]`
Enable/disable spice seamless migration. Default is off.

`gl=[on|off]`
Enable/disable OpenGL context. Default is off.

`-portrait`
Rotate graphical output 90 deg left (only PXA LCD).

`-rotate` *deg*
Rotate graphical output some deg left (only PXA LCD).

`-vga` *type* Select type of VGA card to emulate. Valid values for *type* are

`cirrus` Cirrus Logic GD5446 Video card. All Windows versions starting from Windows 95 should recognize and use this graphic card. For optimal performances, use 16 bit color depth in the guest and the host OS. (This one is the default)

`std` Standard VGA card with Bochs VBE extensions. If your guest OS supports the VESA 2.0 VBE extensions (e.g. Windows XP) and if you want to use high resolution modes (>= 1280x1024x16) then you should use this option.

`vmware` VMWare SVGA-II compatible adapter. Use it if you have sufficiently recent XFree86/XOrg server or Windows guest with a driver for this card.

`qxl` QXL paravirtual graphic card. It is VGA compatible (including VESA 2.0 VBE support). Works best with qxl guest drivers installed though. Recommended choice when using the spice protocol.

`tcx` (sun4m only) Sun TCX framebuffer. This is the default framebuffer for sun4m machines and offers both 8-bit and 24-bit colour depths at a fixed resolution of 1024x768.

`cg3` (sun4m only) Sun cgthree framebuffer. This is a simple 8-bit framebuffer for sun4m machines available in both 1024x768 (OpenBIOS) and 1152x900 (OBP) resolutions aimed at people wishing to run older Solaris versions.

`virtio` Virtio VGA card.

`none` Disable VGA card.

`-full-screen`
Start in full screen.

`-g` *`widthxheight`*`[x`*`depth`*`]`
Set the initial graphical resolution and depth (PPC, SPARC only).

`-vnc` *`display`*`[,`*`option`*`[,`*`option`*`[,...]]]`
Normally, if QEMU is compiled with graphical window support, it displays output such as guest graphics, guest console, and the QEMU monitor in a window. With this option, you can have QEMU listen on VNC display *display* and redirect the VGA display over the VNC session. It is very useful to enable the usb tablet device when using this option (option `-usbdevice tablet`). When using the VNC display, you must use the `-k` parameter to set the keyboard layout if you are not using en-us. Valid syntax for the *display* is

`to=`*`L`*

With this option, QEMU will try next available VNC *display*s, until the number *L*, if the origianlly defined "-vnc *display*" is not available, e.g. port 5900+*display* is already used by another application. By default, to=0.

`host``:`*`d`*

TCP connections will only be allowed from *host* on display *d*. By convention the TCP port is 5900+*d*. Optionally, *host* can be omitted in which case the server will accept connections from any host.

`unix:`*`path`*

Connections will be allowed over UNIX domain sockets where *path* is the location of a unix socket to listen for connections on.

`none`

VNC is initialized but not started. The monitor `change` command can be used to later start the VNC server.

Following the *display* value there may be one or more *option* flags separated by commas. Valid options are

`reverse`

Connect to a listening VNC client via a "reverse" connection. The client is specified by the *display*. For reverse network connections

(*host*:*d*,`reverse`), the *d* argument is a TCP port number, not a display number.

`websocket`

Opens an additional TCP listening port dedicated to VNC Websocket connections. By definition the Websocket port is 5700+*display*. If *host* is specified connections will only be allowed from this host. As an alternative the Websocket port could be specified by using `websocket`=*port*. If no TLS credentials are provided, the websocket connection runs in unencrypted mode. If TLS credentials are provided, the websocket connection requires encrypted client connections.

`password`

Require that password based authentication is used for client connections.

The password must be set separately using the `set_password` command in the Section 2.6 [pcsys_monitor], page 53. The syntax to change your password is: `set_password <protocol> <password>` where <protocol> could be either "vnc" or "spice".

If you would like to change <protocol> password expiration, you should use `expire_password <protocol> <expiration-time>` where expiration time could be one of the following options: now, never, +seconds or UNIX time of expiration, e.g. +60 to make password expire in 60 seconds, or 1335196800 to make password expire on "Mon Apr 23 12:00:00 EDT 2012" (UNIX time for this date and time).

You can also use keywords "now" or "never" for the expiration time to allow <protocol> password to expire immediately or never expire.

`tls-creds=`*ID*

Provides the ID of a set of TLS credentials to use to secure the VNC server. They will apply to both the normal VNC server socket and the websocket socket (if enabled). Setting TLS credentials will cause the VNC server socket to enable the VeNCrypt auth mechanism. The credentials should have been previously created using the `-object tls-creds` argument.

The `tls-creds` parameter obsoletes the `tls`, `x509`, and `x509verify` options, and as such it is not permitted to set both new and old type options at the same time.

`tls`

Require that client use TLS when communicating with the VNC server. This uses anonymous TLS credentials so is susceptible to a man-in-the-middle attack. It is recommended that this option be combined with either the `x509` or `x509verify` options.

This option is now deprecated in favor of using the `tls-creds` argument.

`x509=`*`/path/to/certificate/dir`*

Valid if `tls` is specified. Require that x509 credentials are used for negotiating the TLS session. The server will send its x509 certificate to the client. It is recommended that a password be set on the VNC server to provide authentication of the client when this is used. The path following this option specifies where the x509 certificates are to be loaded from. See the Section 2.12 [vnc_security], page 91, section for details on generating certificates.

This option is now deprecated in favour of using the `tls-creds` argument.

`x509verify=`*`/path/to/certificate/dir`*

Valid if `tls` is specified. Require that x509 credentials are used for negotiating the TLS session. The server will send its x509 certificate to the client, and request that the client send its own x509 certificate. The server will validate the client's certificate against the CA certificate, and reject clients when validation fails. If the certificate authority is trusted, this is a sufficient authentication mechanism. You may still wish to set a password on the VNC server as a second authentication layer. The path following this option specifies where the x509 certificates are to be loaded from. See the Section 2.12 [vnc_security], page 91, section for details on generating certificates.

This option is now deprecated in favour of using the `tls-creds` argument.

`sasl`

Require that the client use SASL to authenticate with the VNC server. The exact choice of authentication method used is controlled from the system / user's SASL configuration file for the 'qemu' service. This is typically found in /etc/sasl2/qemu.conf. If running QEMU as an unprivileged user, an environment variable SASL_CONF_PATH can be used to make it search alternate locations for the service config. While some SASL auth methods can also provide data encryption (eg GSSAPI), it is recommended that SASL always be combined with the 'tls' and 'x509' settings to enable use of SSL and server certificates. This ensures a data encryption preventing compromise of authentication credentials. See the Section 2.12 [vnc_security], page 91, section for details on using SASL authentication.

`acl`

Turn on access control lists for checking of the x509 client certificate and SASL party. For x509 certs, the ACL check is made against the certificate's distinguished name. This is something that looks like

`C=GB,O=ACME,L=Boston,CN=bob`. For SASL party, the ACL check is made against the username, which depending on the SASL plugin, may include a realm component, eg `bob` or `bob@EXAMPLE.COM`. When the `acl` flag is set, the initial access list will be empty, with a `deny` policy. Thus no one will be allowed to use the VNC server until the ACLs have been loaded. This can be achieved using the `acl` monitor command.

`lossy`
: Enable lossy compression methods (gradient, JPEG, ...). If this option is set, VNC client may receive lossy framebuffer updates depending on its encoding settings. Enabling this option can save a lot of bandwidth at the expense of quality.

`non-adaptive`
: Disable adaptive encodings. Adaptive encodings are enabled by default. An adaptive encoding will try to detect frequently updated screen regions, and send updates in these regions using a lossy encoding (like JPEG). This can be really helpful to save bandwidth when playing videos. Disabling adaptive encodings restores the original static behavior of encodings like Tight.

`share=[allow-exclusive|force-shared|ignore]`
: Set display sharing policy. 'allow-exclusive' allows clients to ask for exclusive access. As suggested by the rfb spec this is implemented by dropping other connections. Connecting multiple clients in parallel requires all clients asking for a shared session (vncviewer: -shared switch). This is the default. 'force-shared' disables exclusive client access. Useful for shared desktop sessions, where you don't want someone forgetting specify -shared disconnect everybody else. 'ignore' completely ignores the shared flag and allows everybody connect unconditionally. Doesn't conform to the rfb spec but is traditional QEMU behavior.

`key-delay-ms`
: Set keyboard delay, for key down and key up events, in milliseconds. Default is 1. Keyboards are low-bandwidth devices, so this slowdown can help the device and guest to keep up and not lose events in case events are arriving in bulk. Possible causes for the latter are flaky network connections, or scripts for automated testing.

i386 target only:

`-win2k-hack`
: Use it when installing Windows 2000 to avoid a disk full bug. After Windows 2000 is installed, you no longer need this option (this option slows down the IDE transfers).

`-no-fd-bootchk`
: Disable boot signature checking for floppy disks in BIOS. May be needed to boot from old floppy disks.

`-no-acpi` Disable ACPI (Advanced Configuration and Power Interface) support. Use it if your guest OS complains about ACPI problems (PC target machine only).

`-no-hpet` Disable HPET support.

`-acpitable [sig=`*str*`][,rev=`*n*`][,oem_id=`*str*`][,oem_table_id=`*str*`][,oem_rev=`*n*`]`
`[,asl_compiler_id=`*str*`][,asl_compiler_rev=`*n*`][,data=`*file1*`[:`*file2*`]...]`
Add ACPI table with specified header fields and context from specified files. For file=, take whole ACPI table from the specified files, including all ACPI headers (possible overridden by other options). For data=, only data portion of the table is used, all header information is specified in the command line. If a SLIC table is supplied to QEMU, then the SLIC's oem_id and oem_table_id fields will override the same in the RSDT and the FADT (a.k.a. FACP), in order to ensure the field matches required by the Microsoft SLIC spec and the ACPI spec.

`-smbios file=`*binary*
Load SMBIOS entry from binary file.

`-smbios`
`type=0[,vendor=`*str*`][,version=`*str*`][,date=`*str*`][,release=`*%d.%d*`][,uefi=on|off]`
Specify SMBIOS type 0 fields

`-smbios`
`type=1[,manufacturer=`*str*`][,product=`*str*`][,version=`*str*`][,serial=`*str*`][,uuid=`*uuid*`][,sku=`*str*`][,f`
Specify SMBIOS type 1 fields

`-smbios`
`type=2[,manufacturer=`*str*`][,product=`*str*`][,version=`*str*`][,serial=`*str*`][,asset=`*str*`][,location=`*st*
Specify SMBIOS type 2 fields

`-smbios`
`type=3[,manufacturer=`*str*`][,version=`*str*`][,serial=`*str*`][,asset=`*str*`][,sku=`*str*`]`
Specify SMBIOS type 3 fields

`-smbios type=4[,sock_`
`pfx=`*str*`][,manufacturer=`*str*`][,version=`*str*`][,serial=`*str*`][,asset=`*str*`][,part=`*str*`]`
Specify SMBIOS type 4 fields

`-smbios type=17[,loc_`
`pfx=`*str*`][,bank=`*str*`][,manufacturer=`*str*`][,serial=`*str*`][,asset=`*str*`][,part=`*str*`][,speed=`*%d*`]`
Specify SMBIOS type 17 fields

Network options:

`-net nic[,vlan=`*n*`][,macaddr=`*mac*`][,model=`*type*`]`
`[,name=`*name*`][,addr=`*addr*`][,vectors=`*v*`]`
Create a new Network Interface Card and connect it to VLAN *n* ($n = 0$ is the default). The NIC is an e1000 by default on the PC target. Optionally, the MAC address can be changed to *mac*, the device address set to *addr* (PCI cards only), and a *name* can be assigned for use in monitor commands. Optionally, for PCI cards, you can specify the number *v* of MSI-X vectors that the card should have; this option currently only affects virtio cards; set $v = 0$ to disable

MSI-X. If no `-net` option is specified, a single NIC is created. QEMU can emulate several different models of network card. Valid values for *type* are `virtio`, `i82551`, `i82557b`, `i82559er`, `ne2k_pci`, `ne2k_isa`, `pcnet`, `rtl8139`, `e1000`, `smc91c111`, `lance` and `mcf_fec`. Not all devices are supported on all targets. Use `-net nic,model=help` for a list of available devices for your target.

`-netdev user,id=`*id*`[,`*option*`][,`*option*`][,...]`
`-net user[,`*option*`][,`*option*`][,...]`

Use the user mode network stack which requires no administrator privilege to run. Valid options are:

`vlan=`*n* Connect user mode stack to VLAN *n* (*n* = 0 is the default).

`id=`*id*
`name=`*name*
Assign symbolic name for use in monitor commands.

`ipv4` and `ipv6` specify that either IPv4 or IPv6 must be enabled. If neither is specified both protocols are enabled.

`net=`*addr*`[/`*mask*`]`
Set IP network address the guest will see. Optionally specify the netmask, either in the form a.b.c.d or as number of valid top-most bits. Default is 10.0.2.0/24.

`host=`*addr*
Specify the guest-visible address of the host. Default is the 2nd IP in the guest network, i.e. x.x.x.2.

`ipv6-net=`*addr*`[/`*int*`]`
Set IPv6 network address the guest will see (default is fec0::/64). The network prefix is given in the usual hexadecimal IPv6 address notation. The prefix size is optional, and is given as the number of valid top-most bits (default is 64).

`ipv6-host=`*addr*
Specify the guest-visible IPv6 address of the host. Default is the 2nd IPv6 in the guest network, i.e. xxxx::2.

`restrict=on|off`
If this option is enabled, the guest will be isolated, i.e. it will not be able to contact the host and no guest IP packets will be routed over the host to the outside. This option does not affect any explicitly set forwarding rules.

`hostname=`*name*
Specifies the client hostname reported by the built-in DHCP server.

`dhcpstart=`*addr*
Specify the first of the 16 IPs the built-in DHCP server can assign. Default is the 15th to 31st IP in the guest network, i.e. x.x.x.15 to x.x.x.31.

`dns=`*addr* Specify the guest-visible address of the virtual nameserver. The address must be different from the host address. Default is the 3rd IP in the guest network, i.e. x.x.x.3.

`ipv6-dns=`*addr*
Specify the guest-visible address of the IPv6 virtual nameserver. The address must be different from the host address. Default is the 3rd IP in the guest network, i.e. xxxx::3.

`dnssearch=`*domain*
Provides an entry for the domain-search list sent by the built-in DHCP server. More than one domain suffix can be transmitted by specifying this option multiple times. If supported, this will cause the guest to automatically try to append the given domain suffix(es) in case a domain name can not be resolved.

Example:

```
qemu -net user,dnssearch=mgmt.example.org,dnssearch=example.org [...]
```

`tftp=`*dir* When using the user mode network stack, activate a built-in TFTP server. The files in *dir* will be exposed as the root of a TFTP server. The TFTP client on the guest must be configured in binary mode (use the command `bin` of the Unix TFTP client).

`bootfile=`*file*
When using the user mode network stack, broadcast *file* as the BOOTP filename. In conjunction with `tftp`, this can be used to network boot a guest from a local directory.

Example (using pxelinux):

```
qemu-system-i386 -hda linux.img -boot n -net user,tftp=/path/to/tftp/fi
```

`smb=`*dir*`[,smbserver=`*addr*`]`
When using the user mode network stack, activate a built-in SMB server so that Windows OSes can access to the host files in *dir* transparently. The IP address of the SMB server can be set to *addr*. By default the 4th IP in the guest network is used, i.e. x.x.x.4.

In the guest Windows OS, the line:

```
10.0.2.4 smbserver
```

must be added in the file `C:\WINDOWS\LMHOSTS` (for windows 9x/Me) or `C:\WINNT\SYSTEM32\DRIVERS\ETC\LMHOSTS` (Windows NT/2000).

Then *dir* can be accessed in `\\smbserver\qemu`.

Note that a SAMBA server must be installed on the host OS. QEMU was tested successfully with smbd versions from Red Hat 9, Fedora Core 3 and OpenSUSE 11.x.

`hostfwd=[tcp|udp]:[`*hostaddr*`]:`*hostport*`-[`*guestaddr*`]:`*guestport*
Redirect incoming TCP or UDP connections to the host port *hostport* to the guest IP address *guestaddr* on guest port *guestport*. If

guestaddr is not specified, its value is x.x.x.15 (default first address given by the built-in DHCP server). By specifying *hostaddr*, the rule can be bound to a specific host interface. If no connection type is set, TCP is used. This option can be given multiple times.

For example, to redirect host X11 connection from screen 1 to guest screen 0, use the following:

```
# on the host
qemu-system-i386 -net user,hostfwd=tcp:127.0.0.1:6001-:6000 [...]
# this host xterm should open in the guest X11 server
xterm -display :1
```

To redirect telnet connections from host port 5555 to telnet port on the guest, use the following:

```
# on the host
qemu-system-i386 -net user,hostfwd=tcp::5555-:23 [...]
telnet localhost 5555
```

Then when you use on the host `telnet localhost 5555`, you connect to the guest telnet server.

`guestfwd=[tcp]:`*server*`:`*port*`-`*dev*
`guestfwd=[tcp]:`*server*`:`*port*`-cmd:`*command*

Forward guest TCP connections to the IP address *server* on port *port* to the character device *dev* or to a program executed by *cmd:command* which gets spawned for each connection. This option can be given multiple times.

You can either use a chardev directly and have that one used throughout QEMU's lifetime, like in the following example:

```
# open 10.10.1.1:4321 on bootup, connect 10.0.2.100:1234 to it whenever
# the guest accesses it
qemu -net user,guestfwd=tcp:10.0.2.100:1234-tcp:10.10.1.1:4321 [...]
```

Or you can execute a command on every TCP connection established by the guest, so that QEMU behaves similar to an inetd process for that virtual server:

```
# call "netcat 10.10.1.1 4321" on every TCP connection to 10.0.2.100:12
# and connect the TCP stream to its stdin/stdout
qemu -net 'user,guestfwd=tcp:10.0.2.100:1234-cmd:netcat 10.10.1.1 4321'
```

Note: Legacy stand-alone options -tftp, -bootp, -smb and -redir are still processed and applied to -net user. Mixing them with the new configuration syntax gives undefined results. Their use for new applications is discouraged as they will be removed from future versions.

`-netdev`
`tap,id=`*id*`[,fd=`*h*`][,ifname=`*name*`][,script=`*file*`][,downscript=`*dfile*`][,br=`*bridge*`][,helper=`*helper*`]`
`-net`
`tap[,vlan=`*n*`][,name=`*name*`][,fd=`*h*`][,ifname=`*name*`][,script=`*file*`][,downscript=`*dfile*`][,br=`*bridge*`][`

Connect the host TAP network interface *name* to VLAN *n*.

Use the network script *file* to configure it and the network script *dfile* to deconfigure it. If *name* is not provided, the OS automatically provides one. The default network configure script is `/etc/qemu-ifup` and the default network deconfigure script is `/etc/qemu-ifdown`. Use `script=no` or `downscript=no` to disable script execution.

If running QEMU as an unprivileged user, use the network helper *helper* to configure the TAP interface and attach it to the bridge. The default network helper executable is `/path/to/qemu-bridge-helper` and the default bridge device is `br0`.

`fd`=*h* can be used to specify the handle of an already opened host TAP interface.

Examples:

```
#launch a QEMU instance with the default network script
qemu-system-i386 linux.img -net nic -net tap
```

```
#launch a QEMU instance with two NICs, each one connected
#to a TAP device
qemu-system-i386 linux.img \
-net nic,vlan=0 -net tap,vlan=0,ifname=tap0 \
-net nic,vlan=1 -net tap,vlan=1,ifname=tap1
```

```
#launch a QEMU instance with the default network helper to
#connect a TAP device to bridge br0
qemu-system-i386 linux.img \
-net nic -net tap,"helper=/path/to/qemu-bridge-helper"
```

`-netdev bridge,id=`*id*`[,br=`*bridge*`][,helper=`*helper*`]`
`-net bridge[,vlan=`*n*`][,name=`*name*`][,br=`*bridge*`][,helper=`*helper*`]`

Connect a host TAP network interface to a host bridge device.

Use the network helper *helper* to configure the TAP interface and attach it to the bridge. The default network helper executable is `/path/to/qemu-bridge-helper` and the default bridge device is `br0`.

Examples:

```
#launch a QEMU instance with the default network helper to
#connect a TAP device to bridge br0
qemu-system-i386 linux.img -net bridge -net nic,model=virtio
```

```
#launch a QEMU instance with the default network helper to
#connect a TAP device to bridge qemubr0
qemu-system-i386 linux.img -net bridge,br=qemubr0 -net nic,model=virtio
```

`-netdev socket,id=`*id*`[,fd=`*h*`][,listen=[`*host*`]:`*port*`][,connect=`*host*`:`*port*`]`
`-net socket[,vlan=`*n*`][,name=`*name*`][,fd=`*h*`]`
`[,listen=[`*host*`]:`*port*`][,connect=`*host*`:`*port*`]`

Connect the VLAN *n* to a remote VLAN in another QEMU virtual machine using a TCP socket connection. If `listen` is specified, QEMU waits for incoming connections on *port* (*host* is optional). `connect` is used to connect to another QEMU instance using the `listen` option. `fd`=*h* specifies an already opened TCP socket.

Example:

```
# launch a first QEMU instance
qemu-system-i386 linux.img \
-net nic,macaddr=52:54:00:12:34:56 \
-net socket,listen=:1234
# connect the VLAN 0 of this instance to the VLAN 0
# of the first instance
qemu-system-i386 linux.img \
-net nic,macaddr=52:54:00:12:34:57 \
-net socket,connect=127.0.0.1:1234
```

`-netdev socket,id=`*id*`[,fd=`*h*`][,mcast=`*maddr*`:`*port*`[,localaddr=`*addr*`]]`
`-net socket[,vlan=`*n*`][,name=`*name*`][,fd=`*h*`][,mcast=`*maddr*`:`*port*`[,localaddr=`*addr*`]]`

Create a VLAN *n* shared with another QEMU virtual machines using a UDP multicast socket, effectively making a bus for every QEMU with same multicast address *maddr* and *port*. NOTES:

1. Several QEMU can be running on different hosts and share same bus (assuming correct multicast setup for these hosts).
2. mcast support is compatible with User Mode Linux (argument `ethN=mcast`), see `http://user-mode-linux.sf.net`.
3. Use `fd=h` to specify an already opened UDP multicast socket.

Example:

```
# launch one QEMU instance
qemu-system-i386 linux.img \
-net nic,macaddr=52:54:00:12:34:56 \
-net socket,mcast=230.0.0.1:1234
# launch another QEMU instance on same "bus"
qemu-system-i386 linux.img \
-net nic,macaddr=52:54:00:12:34:57 \
-net socket,mcast=230.0.0.1:1234
# launch yet another QEMU instance on same "bus"
qemu-system-i386 linux.img \
-net nic,macaddr=52:54:00:12:34:58 \
-net socket,mcast=230.0.0.1:1234
```

Example (User Mode Linux compat.):

```
# launch QEMU instance (note mcast address selected
# is UML's default)
qemu-system-i386 linux.img \
-net nic,macaddr=52:54:00:12:34:56 \
-net socket,mcast=239.192.168.1:1102
# launch UML
/path/to/linux ubd0=/path/to/root_fs eth0=mcast
```

Example (send packets from host's 1.2.3.4):

```
qemu-system-i386 linux.img \
-net nic,macaddr=52:54:00:12:34:56 \
```

```
-net socket,mcast=239.192.168.1:1102,localaddr=1.2.3.4
```

`-netdev l2tpv3,id=`*id*`,src=`*srcaddr*`,dst=`*dstaddr*`[,srcport=`*srcport*`][,dstport=`*dstport*`],txsession=`*txsessio*
`-net l2tpv3[,vlan=`*n*`][,name=`*name*`],src=`*srcaddr*`,dst=`*dstaddr*`[,srcport=`*srcport*`][,dstport=`*dstport*`],txs`

Connect VLAN *n* to L2TPv3 pseudowire. L2TPv3 (RFC3391) is a popular protocol to transport Ethernet (and other Layer 2) data frames between two systems. It is present in routers, firewalls and the Linux kernel (from version 3.3 onwards).

This transport allows a VM to communicate to another VM, router or firewall directly.

`src=`*srcaddr*
source address (mandatory)

`dst=`*dstaddr*
destination address (mandatory)

`udp` select udp encapsulation (default is ip).

`srcport=`*srcport*
source udp port.

`dstport=`*dstport*
destination udp port.

`ipv6` force v6, otherwise defaults to v4.

`rxcookie=`*rxcookie*
`txcookie=`*txcookie*
Cookies are a weak form of security in the l2tpv3 specification. Their function is mostly to prevent misconfiguration. By default they are 32 bit.

`cookie64` Set cookie size to 64 bit instead of the default 32

`counter=off`
Force a 'cut-down' L2TPv3 with no counter as in draft-mkonstan-l2tpext-keyed-ipv6-tunnel-00

`pincounter=on`
Work around broken counter handling in peer. This may also help on networks which have packet reorder.

`offset=`*offset*
Add an extra offset between header and data

For example, to attach a VM running on host 4.3.2.1 via L2TPv3 to the bridge br-lan on the remote Linux host 1.2.3.4:

```
# Setup tunnel on linux host using raw ip as encapsulation
# on 1.2.3.4
ip l2tp add tunnel remote 4.3.2.1 local 1.2.3.4 tunnel_id 1 peer_tunnel_id 1 \
encap udp udp_sport 16384 udp_dport 16384
ip l2tp add session tunnel_id 1 name vmtunnel0 session_id \
```

```
0xFFFFFFFF peer_session_id 0xFFFFFFFF
ifconfig vmtunnel0 mtu 1500
ifconfig vmtunnel0 up
brctl addif br-lan vmtunnel0


# on 4.3.2.1
# launch QEMU instance - if your network has reorder or is very lossy add ,pincou

qemu-system-i386 linux.img -net nic -net l2tpv3,src=4.2.3.1,dst=1.2.3.4,udp,srcpo
```

`-netdev`
`vde,id=`*id*`[,sock=`*socketpath*`][,port=`*n*`][,group=`*groupname*`][,mode=`*octalmode*`]`
`-net vde[,vlan=`*n*`][,name=`*name*`][,sock=`*socketpath*`]`
`[,port=`*n*`][,group=`*groupname*`][,mode=`*octalmode*`]`

Connect VLAN *n* to PORT *n* of a vde switch running on host and listening for incoming connections on *socketpath*. Use GROUP *groupname* and MODE *octalmode* to change default ownership and permissions for communication port. This option is only available if QEMU has been compiled with vde support enabled.

Example:

```
# launch vde switch
vde_switch -F -sock /tmp/myswitch
# launch QEMU instance
qemu-system-i386 linux.img -net nic -net vde,sock=/tmp/myswitch
```

`-netdev hubport,id=`*id*`,hubid=`*hubid*

Create a hub port on QEMU "vlan" *hubid*.

The hubport netdev lets you connect a NIC to a QEMU "vlan" instead of a single netdev. `-net` and `-device` with parameter `vlan` create the required hub automatically.

`-netdev vhost-user,chardev=`*id*`[,vhostforce=on|off][,queues=n]`

Establish a vhost-user netdev, backed by a chardev *id*. The chardev should be a unix domain socket backed one. The vhost-user uses a specifically defined protocol to pass vhost ioctl replacement messages to an application on the other end of the socket. On non-MSIX guests, the feature can be forced with *vhostforce*. Use 'queues=*n*' to specify the number of queues to be created for multiqueue vhost-user.

Example:

```
qemu -m 512 -object memory-backend-file,id=mem,size=512M,mem-path=/hugetlbfs,shar
-numa node,memdev=mem \
-chardev socket,path=/path/to/socket \
-netdev type=vhost-user,id=net0,chardev=chr0 \
-device virtio-net-pci,netdev=net0
```

`-net dump[,vlan=n][,file=file][,len=len]`
: Dump network traffic on VLAN *n* to file *file* (`qemu-vlan0.pcap` by default). At most *len* bytes (64k by default) per packet are stored. The file format is libpcap, so it can be analyzed with tools such as tcpdump or Wireshark. Note: For devices created with '-netdev', use '-object filter-dump,...' instead.

`-net none`
: Indicate that no network devices should be configured. It is used to override the default configuration (`-net nic -net user`) which is activated if no `-net` options are provided.

Character device options:

The general form of a character device option is:

`-chardev backend ,id=id [,mux=on|off] [,options]`
: Backend is one of: `null`, `socket`, `udp`, `msmouse`, `vc`, `ringbuf`, `file`, `pipe`, `console`, `serial`, `pty`, `stdio`, `braille`, `tty`, `parallel`, `parport`, `spicevmc`. `spiceport`. The specific backend will determine the applicable options.

  Use "-chardev help" to print all available chardev backend types.

  All devices must have an id, which can be any string up to 127 characters long. It is used to uniquely identify this device in other command line directives.

  A character device may be used in multiplexing mode by multiple front-ends. Specify `mux=on` to enable this mode. A multiplexer is a "1:N" device, and here the "1" end is your specified chardev backend, and the "N" end is the various parts of QEMU that can talk to a chardev. If you create a chardev with `id=myid` and `mux=on`, QEMU will create a multiplexer with your specified ID, and you can then configure multiple front ends to use that chardev ID for their input/output. Up to four different front ends can be connected to a single multiplexed chardev. (Without multiplexing enabled, a chardev can only be used by a single front end.) For instance you could use this to allow a single stdio chardev to be used by two serial ports and the QEMU monitor:

```
-chardev stdio,mux=on,id=char0 \
-mon chardev=char0,mode=readline \
-serial chardev:char0 \
-serial chardev:char0
```

  You can have more than one multiplexer in a system configuration; for instance you could have a TCP port multiplexed between UART 0 and UART 1, and stdio multiplexed between the QEMU monitor and a parallel port:

```
-chardev stdio,mux=on,id=char0 \
-mon chardev=char0,mode=readline \
-parallel chardev:char0 \
-chardev tcp,...,mux=on,id=char1 \
-serial chardev:char1 \
-serial chardev:char1
```

  When you're using a multiplexed character device, some escape sequences are interpreted in the input. See Section 2.5 [mux_keys], page 52.

  Note that some other command line options may implicitly create multiplexed character backends; for instance `-serial mon:stdio` creates a multiplexed stdio

backend connected to the serial port and the QEMU monitor, and `-nographic` also multiplexes the console and the monitor to stdio.

There is currently no support for multiplexing in the other direction (where a single QEMU front end takes input and output from multiple chardevs).

Every backend supports the `logfile` option, which supplies the path to a file to record all data transmitted via the backend. The `logappend` option controls whether the log file will be truncated or appended to when opened.

Further options to each backend are described below.

`-chardev null ,id=`*id*

A void device. This device will not emit any data, and will drop any data it receives. The null backend does not take any options.

`-chardev socket ,id=`*id* `[`*TCP options* `or` *unix options*`] [,server] [,nowait] [,telnet] [,reconnect=`*seconds*`] [,tls-creds=`*id*`]`

Create a two-way stream socket, which can be either a TCP or a unix socket. A unix socket will be created if `path` is specified. Behaviour is undefined if TCP options are specified for a unix socket.

`server` specifies that the socket shall be a listening socket.

`nowait` specifies that QEMU should not block waiting for a client to connect to a listening socket.

`telnet` specifies that traffic on the socket should interpret telnet escape sequences.

`reconnect` sets the timeout for reconnecting on non-server sockets when the remote end goes away. qemu will delay this many seconds and then attempt to reconnect. Zero disables reconnecting, and is the default.

`tls-creds` requests enablement of the TLS protocol for encryption, and specifies the id of the TLS credentials to use for the handshake. The credentials must be previously created with the `-object tls-creds` argument.

TCP and unix socket options are given below:

`TCP options: port=`*port* `[,host=`*host*`] [,to=`*to*`] [,ipv4] [,ipv6] [,nodelay]`

`host` for a listening socket specifies the local address to be bound. For a connecting socket species the remote host to connect to. `host` is optional for listening sockets. If not specified it defaults to `0.0.0.0`.

`port` for a listening socket specifies the local port to be bound. For a connecting socket specifies the port on the remote host to connect to. `port` can be given as either a port number or a service name. `port` is required.

`to` is only relevant to listening sockets. If it is specified, and `port` cannot be bound, QEMU will attempt to bind to subsequent ports up to and including `to` until it succeeds. `to` must be specified as a port number.

`ipv4` and `ipv6` specify that either IPv4 or IPv6 must be used. If neither is specified the socket may use either protocol.

`nodelay` disables the Nagle algorithm.

`unix options: path=`*`path`*

`path` specifies the local path of the unix socket. `path` is required.

`-chardev udp ,id=`*`id`* `[,host=`*`host`*`] ,port=`*`port`* `[,localaddr=`*`localaddr`*`] [,localport=`*`localport`*`] [,ipv4] [,ipv6]`

Sends all traffic from the guest to a remote host over UDP.

`host` specifies the remote host to connect to. If not specified it defaults to `localhost`.

`port` specifies the port on the remote host to connect to. `port` is required.

`localaddr` specifies the local address to bind to. If not specified it defaults to `0.0.0.0`.

`localport` specifies the local port to bind to. If not specified any available local port will be used.

`ipv4` and `ipv6` specify that either IPv4 or IPv6 must be used. If neither is specified the device may use either protocol.

`-chardev msmouse ,id=`*`id`*

Forward QEMU's emulated msmouse events to the guest. `msmouse` does not take any options.

`-chardev vc ,id=`*`id`* `[[,width=`*`width`*`] [,height=`*`height`*`]] [[,cols=`*`cols`*`] [,rows=`*`rows`*`]]`

Connect to a QEMU text console. `vc` may optionally be given a specific size.

`width` and `height` specify the width and height respectively of the console, in pixels.

`cols` and `rows` specify that the console be sized to fit a text console with the given dimensions.

`-chardev ringbuf ,id=`*`id`* `[,size=`*`size`*`]`

Create a ring buffer with fixed size `size`. *size* must be a power of two and defaults to `64K`.

`-chardev file ,id=`*`id`* `,path=`*`path`*

Log all traffic received from the guest to a file.

`path` specifies the path of the file to be opened. This file will be created if it does not already exist, and overwritten if it does. `path` is required.

`-chardev pipe ,id=`*`id`* `,path=`*`path`*

Create a two-way connection to the guest. The behaviour differs slightly between Windows hosts and other hosts:

On Windows, a single duplex pipe will be created at `\\.pipe\path`.

On other hosts, 2 pipes will be created called `path.in` and `path.out`. Data written to `path.in` will be received by the guest. Data written by the guest can be read from `path.out`. QEMU will not create these fifos, and requires them to be present.

`path` forms part of the pipe path as described above. `path` is required.

`-chardev console ,id=id`
: Send traffic from the guest to QEMU's standard output. `console` does not take any options.

  `console` is only available on Windows hosts.

`-chardev serial ,id=id ,path=path`
: Send traffic from the guest to a serial device on the host.

  On Unix hosts serial will actually accept any tty device, not only serial lines.

  `path` specifies the name of the serial device to open.

`-chardev pty ,id=id`
: Create a new pseudo-terminal on the host and connect to it. `pty` does not take any options.

  `pty` is not available on Windows hosts.

`-chardev stdio ,id=id [,signal=on|off]`
: Connect to standard input and standard output of the QEMU process.

  `signal` controls if signals are enabled on the terminal, that includes exiting QEMU with the key sequence `Control-c`. This option is enabled by default, use `signal=off` to disable it.

  `stdio` is not available on Windows hosts.

`-chardev braille ,id=id`
: Connect to a local BrlAPI server. `braille` does not take any options.

`-chardev tty ,id=id ,path=path`
: `tty` is only available on Linux, Sun, FreeBSD, NetBSD, OpenBSD and DragonFlyBSD hosts. It is an alias for `serial`.

  `path` specifies the path to the tty. `path` is required.

`-chardev parallel ,id=id ,path=path`
`-chardev parport ,id=id ,path=path`
: `parallel` is only available on Linux, FreeBSD and DragonFlyBSD hosts.

  Connect to a local parallel port.

  `path` specifies the path to the parallel port device. `path` is required.

`-chardev spicevmc ,id=id ,debug=debug, name=name`
: `spicevmc` is only available when spice support is built in.

  `debug` debug level for spicevmc

  `name` name of spice channel to connect to

  Connect to a spice virtual machine channel, such as vdiport.

`-chardev spiceport ,id=id ,debug=debug, name=name`
: `spiceport` is only available when spice support is built in.

  `debug` debug level for spicevmc

  `name` name of spice port to connect to

  Connect to a spice port, allowing a Spice client to handle the traffic identified by a name (preferably a fqdn).

Device URL Syntax:

In addition to using normal file images for the emulated storage devices, QEMU can also use networked resources such as iSCSI devices. These are specified using a special URL syntax.

`iSCSI` iSCSI support allows QEMU to access iSCSI resources directly and use as images for the guest storage. Both disk and cdrom images are supported.

Syntax for specifying iSCSI LUNs is "iscsi://<target-ip>[:<port>]/<target-iqn>/<lun>"

By default qemu will use the iSCSI initiator-name 'iqn.2008-11.org.linux-kvm[:<name>]' but this can also be set from the command line or a configuration file.

Since version Qemu 2.4 it is possible to specify a iSCSI request timeout to detect stalled requests and force a reestablishment of the session. The timeout is specified in seconds. The default is 0 which means no timeout. Libiscsi 1.15.0 or greater is required for this feature.

Example (without authentication):

```
qemu-system-i386 -iscsi initiator-name=iqn.2001-04.com.example:my-initiator \
-cdrom iscsi://192.0.2.1/iqn.2001-04.com.example/2 \
-drive file=iscsi://192.0.2.1/iqn.2001-04.com.example/1
```

Example (CHAP username/password via URL):

```
qemu-system-i386 -drive file=iscsi://user%password@192.0.2.1/iqn.2001-04.com.exam
```

Example (CHAP username/password via environment variables):

```
LIBISCSI_CHAP_USERNAME="user" \
LIBISCSI_CHAP_PASSWORD="password" \
qemu-system-i386 -drive file=iscsi://192.0.2.1/iqn.2001-04.com.example/1
```

iSCSI support is an optional feature of QEMU and only available when compiled and linked against libiscsi.

iSCSI parameters such as username and password can also be specified via a configuration file. See qemu-doc for more information and examples.

`NBD` QEMU supports NBD (Network Block Devices) both using TCP protocol as well as Unix Domain Sockets.

Syntax for specifying a NBD device using TCP "nbd:<server-ip>:<port>[:exportname=<export>]"

Syntax for specifying a NBD device using Unix Domain Sockets "nbd:unix:<domain-socket>[:exportname=<export>]"

Example for TCP

```
qemu-system-i386 --drive file=nbd:192.0.2.1:30000
```

Example for Unix Domain Sockets

```
qemu-system-i386 --drive file=nbd:unix:/tmp/nbd-socket
```

`SSH` QEMU supports SSH (Secure Shell) access to remote disks.

Examples:

```
qemu-system-i386 -drive file=ssh://user@host/path/to/disk.img
```

```
qemu-system-i386 -drive file.driver=ssh,file.user=user,file.host=host,file.port=2
```

Currently authentication must be done using ssh-agent. Other authentication methods may be supported in future.

`Sheepdog` Sheepdog is a distributed storage system for QEMU. QEMU supports using either local sheepdog devices or remote networked devices.

Syntax for specifying a sheepdog device

```
sheepdog[+tcp|+unix]://[host:port]/vdiname[?socket=path][#snapid|#tag]
```

Example

```
qemu-system-i386 --drive file=sheepdog://192.0.2.1:30000/MyVirtualMachine
```

See also `http://http://www.osrg.net/sheepdog/`.

`GlusterFS`

GlusterFS is an user space distributed file system. QEMU supports the use of GlusterFS volumes for hosting VM disk images using TCP, Unix Domain Sockets and RDMA transport protocols.

Syntax for specifying a VM disk image on GlusterFS volume is

```
URI:
gluster[+type]://[host[:port]]/volume/path[?socket=...][,debug=N][,logfile=...]

JSON:
'json:{"driver":"qcow2","file":{"driver":"gluster","volume":"testvol","path":"a.i
                                  "server":[{"type":"tcp","host":"...","port":"...
                                            {"type":"unix","socket":"..."}]}}'
```

Example

```
URI:
qemu-system-x86_64 --drive file=gluster://192.0.2.1/testvol/a.img,
                                file.debug=9,file.logfile=/var/log/qemu-gluster.lo

JSON:
qemu-system-x86_64 'json:{"driver":"qcow2",
                          "file":{"driver":"gluster",
                                  "volume":"testvol","path":"a.img",
                                  "debug":9,"logfile":"/var/log/qemu-gluster.log
                                  "server":[{"type":"tcp","host":"1.2.3.4","port
                                            {"type":"unix","socket":"/var/run/gl
qemu-system-x86_64 -drive driver=qcow2,file.driver=gluster,file.volume=testvol,fi
                                     file.debug=9,file.logfile=/var/log/qemu-glu
                                     file.server.0.type=tcp,file.server.0.host=1
                                     file.server.1.type=unix,file.server.1.socke
```

See also `http://www.gluster.org`.

`HTTP/HTTPS/FTP/FTPS`

QEMU supports read-only access to files accessed over http(s) and ftp(s).

Syntax using a single filename:

`<protocol>://[<username>[:<password>]@]<host>/<path>`

where:

`protocol` 'http', 'https', 'ftp', or 'ftps'.

`username` Optional username for authentication to the remote server.

`password` Optional password for authentication to the remote server.

`host` Address of the remote server.

`path` Path on the remote server, including any query string.

The following options are also supported:

`url` The full URL when passing options to the driver explicitly.

`readahead`
The amount of data to read ahead with each range request to the remote server. This value may optionally have the suffix 'T', 'G', 'M', 'K', 'k' or 'b'. If it does not have a suffix, it will be assumed to be in bytes. The value must be a multiple of 512 bytes. It defaults to 256k.

`sslverify`
Whether to verify the remote server's certificate when connecting over SSL. It can have the value 'on' or 'off'. It defaults to 'on'.

`cookie` Send this cookie (it can also be a list of cookies separated by ';') with each outgoing request. Only supported when using protocols such as HTTP which support cookies, otherwise ignored.

`timeout` Set the timeout in seconds of the CURL connection. This timeout is the time that CURL waits for a response from the remote server to get the size of the image to be downloaded. If not set, the default timeout of 5 seconds is used.

Note that when passing options to qemu explicitly, `driver` is the value of <protocol>.

Example: boot from a remote Fedora 20 live ISO image

```
qemu-system-x86_64 --drive media=cdrom,file=http://dl.fedoraproject.org/pub/fedor

qemu-system-x86_64 --drive media=cdrom,file.driver=http,file.url=http://dl.fedora
```

Example: boot from a remote Fedora 20 cloud image using a local overlay for writes, copy-on-read, and a readahead of 64k

```
qemu-img create -f qcow2 -o backing_file='json:{"file.driver":"http",, "file.url"

qemu-system-x86_64 -drive file=/tmp/Fedora-x86_64-20-20131211.1-sda.qcow2,copy-on
```

Example: boot from an image stored on a VMware vSphere server with a self-signed certificate using a local overlay for writes, a readahead of 64k and a timeout of 10 seconds.

```
qemu-img create -f qcow2 -o backing_file='json:{"file.driver":"https",, "file.url
```

```
qemu-system-x86_64 -drive file=/tmp/test.qcow2
```

Bluetooth(R) options:

`-bt hci[...]`
: Defines the function of the corresponding Bluetooth HCI. -bt options are matched with the HCIs present in the chosen machine type. For example when emulating a machine with only one HCI built into it, only the first `-bt hci[...]` option is valid and defines the HCI's logic. The Transport Layer is decided by the machine type. Currently the machines `n800` and `n810` have one HCI and all other machines have none.

  The following three types are recognized:

  `-bt hci,null`
  : (default) The corresponding Bluetooth HCI assumes no internal logic and will not respond to any HCI commands or emit events.

  `-bt hci,host[:id]`
  : (`bluez` only) The corresponding HCI passes commands / events to / from the physical HCI identified by the name *id* (default: `hci0`) on the computer running QEMU. Only available on `bluez` capable systems like Linux.

  `-bt hci[,vlan=n]`
  : Add a virtual, standard HCI that will participate in the Bluetooth scatternet *n* (default `0`). Similarly to `-net` VLANs, devices inside a bluetooth network *n* can only communicate with other devices in the same network (scatternet).

`-bt vhci[,vlan=n]`
: (Linux-host only) Create a HCI in scatternet *n* (default 0) attached to the host bluetooth stack instead of to the emulated target. This allows the host and target machines to participate in a common scatternet and communicate. Requires the Linux `vhci` driver installed. Can be used as following:

  ```
  qemu-system-i386 [...OPTIONS...] -bt hci,vlan=5 -bt vhci,vlan=5
  ```

`-bt device:dev[,vlan=n]`
: Emulate a bluetooth device *dev* and place it in network *n* (default `0`). QEMU can only emulate one type of bluetooth devices currently:

  `keyboard` Virtual wireless keyboard implementing the HIDP bluetooth profile.

TPM device options:

The general form of a TPM device option is:

`-tpmdev backend ,id=id [,options]`
: Backend type must be: `passthrough`.

  The specific backend type will determine the applicable options. The `-tpmdev` option creates the TPM backend and requires a `-device` option that specifies the TPM frontend interface model.

Options to each backend are described below.

Use 'help' to print all available TPM backend types.

```
qemu -tpmdev help
```

`-tpmdev passthrough, id=`*id*`, path=`*path*`, cancel-path=`*cancel-path*

(Linux-host only) Enable access to the host's TPM using the passthrough driver.

`path` specifies the path to the host's TPM device, i.e., on a Linux host this would be `/dev/tpm0`. `path` is optional and by default `/dev/tpm0` is used.

`cancel-path` specifies the path to the host TPM device's sysfs entry allowing for cancellation of an ongoing TPM command. `cancel-path` is optional and by default QEMU will search for the sysfs entry to use.

Some notes about using the host's TPM with the passthrough driver:

The TPM device accessed by the passthrough driver must not be used by any other application on the host.

Since the host's firmware (BIOS/UEFI) has already initialized the TPM, the VM's firmware (BIOS/UEFI) will not be able to initialize the TPM again and may therefore not show a TPM-specific menu that would otherwise allow the user to configure the TPM, e.g., allow the user to enable/disable or activate/deactivate the TPM. Further, if TPM ownership is released from within a VM then the host's TPM will get disabled and deactivated. To enable and activate the TPM again afterwards, the host has to be rebooted and the user is required to enter the firmware's menu to enable and activate the TPM. If the TPM is left disabled and/or deactivated most TPM commands will fail.

To create a passthrough TPM use the following two options:

```
-tpmdev passthrough,id=tpm0 -device tpm-tis,tpmdev=tpm0
```

Note that the `-tpmdev` id is `tpm0` and is referenced by `tpmdev=tpm0` in the device option.

Linux/Multiboot boot specific:

When using these options, you can use a given Linux or Multiboot kernel without installing it in the disk image. It can be useful for easier testing of various kernels.

`-kernel` *bzImage*

Use *bzImage* as kernel image. The kernel can be either a Linux kernel or in multiboot format.

`-append` *cmdline*

Use *cmdline* as kernel command line

`-initrd` *file*

Use *file* as initial ram disk.

`-initrd "`*file1* `arg=foo,`*file2*`"`

This syntax is only available with multiboot.

Use *file1* and *file2* as modules and pass arg=foo as parameter to the first module.

`-dtb` *file* Use *file* as a device tree binary (dtb) image and pass it to the kernel on boot.

Microcontroller/Cortex-M specific:

Unlike Linux machines, microcontrollers like Cortex-M MCUs, do not boot from a device; they have the application code written into flash and at reset they directly start executing it. Therefore there is no need to specify -kernel/-initrd, and a simpler solution is available to define the image file that QEMU will use as flash content to execute.

`-image` *elf-file*

Use *elf-file* sections as image of the application to emulate. It is the same file used to program the flash via a JTAG/SWD programmer.

`-board [type=]`*name*

Select the emulated board by *name*. Use `-board help` to list available boards. The names generally follow the CMSIS board definitions and case is important. Each board defines a certain MCU, but a different MCU can be used during emulation if `-mcu` is added.

If not specified, a default board is used, and `-mcu` becomes mandatory.

`-mcu` *model*

Select MCU model (`-mcu help` for list and additional feature selection). The names follow the CMSIS device definitions and case is significant.

If not specified, the board default is used.

Debug/Expert options:

`-fw_cfg [name=]`*name*`,file=`*file*

Add named fw_cfg entry with contents from file *file*.

`-fw_cfg [name=]`*name*`,string=`*str*

Add named fw_cfg entry with contents from string *str*.

The terminating NUL character of the contents of *str* will not be included as part of the fw_cfg item data. To insert contents with embedded NUL characters, you have to use the *file* parameter.

The fw_cfg entries are passed by QEMU through to the guest.

Example:

```
-fw_cfg name=opt/com.mycompany/blob,file=./my_blob.bin
```

creates an fw_cfg entry named opt/com.mycompany/blob with contents from ./my_blob.bin.

`-serial` *dev*

Redirect the virtual serial port to host character device *dev*. The default device is `vc` in graphical mode and `stdio` in non graphical mode.

This option can be used several times to simulate up to 4 serial ports.

Use `-serial none` to disable all serial ports.

Available character devices are:

`vc[:`*WxH*`]` Virtual console. Optionally, a width and height can be given in pixel with

```
vc:800x600
```

It is also possible to specify width or height in characters:

```
vc:80Cx24C
```

`pty` [Linux only] Pseudo TTY (a new PTY is automatically allocated)

`none` No device is allocated.

`null` void device

`chardev:`*id*
Use a named character device defined with the `-chardev` option.

`/dev/XXX` [Linux only] Use host tty, e.g. `/dev/ttyS0`. The host serial port parameters are set according to the emulated ones.

`/dev/parport`*N*
[Linux only, parallel port only] Use host parallel port *N*. Currently SPP and EPP parallel port features can be used.

`file:`*filename*
Write output to *filename*. No character can be read.

`stdio` [Unix only] standard input/output

`pipe:`*filename*
name pipe *filename*

`COM`*n* [Windows only] Use host serial port *n*

`udp:[`*remote_host*`]:`*remote_port*`[@[`*src_ip*`]:`*src_port*`]`
This implements UDP Net Console. When *remote_host* or *src_ip* are not specified they default to `0.0.0.0`. When not using a specified *src_port* a random port is automatically chosen.

If you just want a simple readonly console you can use `netcat` or `nc`, by starting QEMU with: `-serial udp::4555` and nc as: `nc -u -l -p 4555`. Any time QEMU writes something to that port it will appear in the netconsole session.

If you plan to send characters back via netconsole or you want to stop and start QEMU a lot of times, you should have QEMU use the same source port each time by using something like `-serial udp::4555@:4556` to QEMU. Another approach is to use a patched version of netcat which can listen to a TCP port and send and receive characters via udp. If you have a patched version of netcat which activates telnet remote echo and single char transfer, then you can use the following options to step up a netcat redirector to allow telnet on port 5555 to access the QEMU port.

`QEMU Options:`
-serial udp::4555@:4556

`netcat options:`
-u -P 4555 -L 0.0.0.0:4556 -t -p 5555 -I -T

`telnet options:`
localhost 5555

`tcp:[host]:port[,server][,nowait][,nodelay][,reconnect=seconds]`
The TCP Net Console has two modes of operation. It can send the serial I/O to a location or wait for a connection from a location. By default the TCP Net Console is sent to *host* at the *port*. If you use the *server* option QEMU will wait for a client socket application to connect to the port before continuing, unless the `nowait` option was specified. The `nodelay` option disables the Nagle buffering algorithm. The `reconnect` option only applies if *noserver* is set, if the connection goes down it will attempt to reconnect at the given interval. If *host* is omitted, 0.0.0.0 is assumed. Only one TCP connection at a time is accepted. You can use `telnet` to connect to the corresponding character device.

`Example to send tcp console to 192.168.0.2 port 4444`
-serial tcp:192.168.0.2:4444

`Example to listen and wait on port 4444 for connection`
-serial tcp::4444,server

`Example to not wait and listen on ip 192.168.0.100 port 4444`
-serial tcp:192.168.0.100:4444,server,nowait

`telnet:host:port[,server][,nowait][,nodelay]`
The telnet protocol is used instead of raw tcp sockets. The options work the same as if you had specified `-serial tcp`. The difference is that the port acts like a telnet server or client using telnet option negotiation. This will also allow you to send the MAGIC_SYSRQ sequence if you use a telnet that supports sending the break sequence. Typically in unix telnet you do it with Control-] and then type "send break" followed by pressing the enter key.

`unix:path[,server][,nowait][,reconnect=seconds]`
A unix domain socket is used instead of a tcp socket. The option works the same as if you had specified `-serial tcp` except the unix domain socket *path* is used for connections.

`mon:dev_string`
This is a special option to allow the monitor to be multiplexed onto another serial port. The monitor is accessed with key sequence of `Control-a` and then pressing `c`. *dev_string* should be any one of the serial devices specified above. An example to multiplex the monitor onto a telnet server listening on port 4444 would be:

`-serial mon:telnet::4444,server,nowait`

When the monitor is multiplexed to stdio in this way, Ctrl+C will not terminate QEMU any more but will be passed to the guest instead.

`braille` Braille device. This will use BrlAPI to display the braille output on a real or fake device.

`msmouse` Three button serial mouse. Configure the guest to use Microsoft protocol.

`-parallel` *dev*
Redirect the virtual parallel port to host device *dev* (same devices as the serial port). On Linux hosts, `/dev/parportN` can be used to use hardware devices connected on the corresponding host parallel port.

This option can be used several times to simulate up to 3 parallel ports.

Use `-parallel none` to disable all parallel ports.

`-monitor` *dev*
Redirect the monitor to host device *dev* (same devices as the serial port). The default device is `vc` in graphical mode and `stdio` in non graphical mode. Use `-monitor none` to disable the default monitor.

`-qmp` *dev* Like -monitor but opens in 'control' mode.

`-qmp-pretty` *dev*
Like -qmp but uses pretty JSON formatting.

`-mon [chardev=]name[,mode=readline|control]`
Setup monitor on chardev *name*.

`-debugcon` *dev*
Redirect the debug console to host device *dev* (same devices as the serial port). The debug console is an I/O port which is typically port 0xe9; writing to that I/O port sends output to this device. The default device is `vc` in graphical mode and `stdio` in non graphical mode.

`-pidfile` *file*
Store the QEMU process PID in *file*. It is useful if you launch QEMU from a script.

`-singlestep`
Run the emulation in single step mode.

`-S` Do not start CPU at startup (you must type 'c' in the monitor).

`-realtime mlock=on|off`
Run qemu with realtime features. mlocking qemu and guest memory can be enabled via `mlock=on` (enabled by default).

`-gdb` *dev* Wait for gdb connection on device *dev* (see Section 2.13 [gdb_usage], page 95). Typical connections will likely be TCP-based, but also UDP, pseudo TTY, or even stdio are reasonable use case. The latter is allowing to start QEMU from within gdb and establish the connection via a pipe:

```
(gdb) target remote | exec qemu-system-i386 -gdb stdio ...
```

`-s` Shorthand for -gdb tcp::1234, i.e. open a gdbserver on TCP port 1234 (see Section 2.13 [gdb_usage], page 95).

`-d` *item1*`[,...]`
Enable logging of specified items. Use '-d help' for a list of log items.

`-D` *logfile*
: Output log in *logfile* instead of to stderr

`-dfilter` *range1*`[,...]`
: Filter debug output to that relevant to a range of target addresses. The filter spec can be either *start*+*size*, *start*-*size* or *start*..*end* where *start end* and *size* are the addresses and sizes required. For example:

  `-dfilter 0x8000..0x8fff,0xffffffc000080000+0x200,0xffffffc000060000-0x1000`

  Will dump output for any code in the 0x1000 sized block starting at 0x8000 and the 0x200 sized block starting at 0xffffffc000080000 and another 0x1000 sized block starting at 0xffffffc00005f000.

`-L` *path*
: Set the directory for the BIOS, VGA BIOS and keymaps.

  To list all the data directories, use `-L help`.

`-bios` *file*
: Set the filename for the BIOS.

`-enable-kvm`
: Enable KVM full virtualization support. This option is only available if KVM support is enabled when compiling.

`-xen-domid` *id*
: Specify xen guest domain *id* (XEN only).

`-xen-create`
: Create domain using xen hypercalls, bypassing xend. Warning: should not be used when xend is in use (XEN only).

`-xen-attach`
: Attach to existing xen domain. xend will use this when starting QEMU (XEN only).

`-no-reboot`
: Exit instead of rebooting.

`-no-shutdown`
: Don't exit QEMU on guest shutdown, but instead only stop the emulation. This allows for instance switching to monitor to commit changes to the disk image.

`-loadvm` *file*
: Start right away with a saved state (`loadvm` in monitor)

`-daemonize`
: Daemonize the QEMU process after initialization. QEMU will not detach from standard IO until it is ready to receive connections on any of its devices. This option is a useful way for external programs to launch QEMU without having to cope with initialization race conditions.

`-option-rom` *file*
: Load the contents of *file* as an option ROM. This option is useful to load things like EtherBoot.

`-rtc [base=utc|localtime|`*date*`][,clock=host|vm][,driftfix=none|slew]`
: Specify `base` as `utc` or `localtime` to let the RTC start at the current UTC or local time, respectively. `localtime` is required for correct date in MS-DOS or Windows. To start at a specific point in time, provide *date* in the format `2006-06-17T16:01:21` or `2006-06-17`. The default base is UTC.

  By default the RTC is driven by the host system time. This allows using of the RTC as accurate reference clock inside the guest, specifically if the host time is smoothly following an accurate external reference clock, e.g. via NTP. If you want to isolate the guest time from the host, you can set `clock` to `rt` instead. To even prevent it from progressing during suspension, you can set it to `vm`.

  Enable `driftfix` (i386 targets only) if you experience time drift problems, specifically with Windows' ACPI HAL. This option will try to figure out how many timer interrupts were not processed by the Windows guest and will re-inject them.

`-icount [shift=`*N*`|auto][,rr=record|replay,rrfile=`*filename*`]`
: Enable virtual instruction counter. The virtual cpu will execute one instruction every 2^*N* ns of virtual time. If `auto` is specified then the virtual cpu speed will be automatically adjusted to keep virtual time within a few seconds of real time.

  When the virtual cpu is sleeping, the virtual time will advance at default speed unless `sleep=on|off` is specified. With `sleep=on|off`, the virtual time will jump to the next timer deadline instantly whenever the virtual cpu goes to sleep mode and will not advance if no timer is enabled. This behavior give deterministic execution times from the guest point of view.

  Note that while this option can give deterministic behavior, it does not provide cycle accurate emulation. Modern CPUs contain superscalar out of order cores with complex cache hierarchies. The number of instructions executed often has little or no correlation with actual performance.

  `align=on` will activate the delay algorithm which will try to synchronise the host clock and the virtual clock. The goal is to have a guest running at the real frequency imposed by the shift option. Whenever the guest clock is behind the host clock and if `align=on` is specified then we print a message to the user to inform about the delay. Currently this option does not work when `shift` is `auto`. Note: The sync algorithm will work for those shift values for which the guest clock runs ahead of the host clock. Typically this happens when the shift value is high (how high depends on the host machine).

  When `rr` option is specified deterministic record/replay is enabled. Replay log is written into *filename* file in record mode and read from this file in replay mode.

`-watchdog `*model*
: Create a virtual hardware watchdog device. Once enabled (by a guest action), the watchdog must be periodically polled by an agent inside the guest or else the guest will be restarted. Choose a model for which your guest has drivers.

The *model* is the model of hardware watchdog to emulate. Use `-watchdog help` to list available hardware models. Only one watchdog can be enabled for a guest.

The following models may be available:

`ib700` iBASE 700 is a very simple ISA watchdog with a single timer.

`i6300esb` Intel 6300ESB I/O controller hub is a much more featureful PCI-based dual-timer watchdog.

`diag288` A virtual watchdog for s390x backed by the diagnose 288 hypercall (currently KVM only).

`-watchdog-action` *action*

The *action* controls what QEMU will do when the watchdog timer expires. The default is `reset` (forcefully reset the guest). Other possible actions are: `shutdown` (attempt to gracefully shutdown the guest), `poweroff` (forcefully poweroff the guest), `pause` (pause the guest), `debug` (print a debug message and continue), or `none` (do nothing).

Note that the `shutdown` action requires that the guest responds to ACPI signals, which it may not be able to do in the sort of situations where the watchdog would have expired, and thus `-watchdog-action shutdown` is not recommended for production use.

Examples:

```
-watchdog i6300esb -watchdog-action pause
-watchdog ib700
```

`-echr` *numeric_ascii_value*

Change the escape character used for switching to the monitor when using monitor and serial sharing. The default is `0x01` when using the `-nographic` option. `0x01` is equal to pressing `Control-a`. You can select a different character from the ascii control keys where 1 through 26 map to Control-a through Control-z. For instance you could use the either of the following to change the escape character to Control-t.

```
-echr 0x14
-echr 20
```

`-virtioconsole` *c*

Set virtio console.

This option is maintained for backward compatibility.

Please use `-device virtconsole` for the new way of invocation.

`-show-cursor`

Show cursor.

`-tb-size` *n*

Set TB size.

`-incoming tcp:[`*host*`]:`*port*`[,to=`*maxport*`][,ipv4][,ipv6]`
`-incoming rdma:`*host*`:`*port*`[,ipv4][,ipv6]`

Prepare for incoming migration, listen on a given tcp port.

`-incoming unix:`*`socketpath`*
: Prepare for incoming migration, listen on a given unix socket.

`-incoming fd:`*`fd`*
: Accept incoming migration from a given filedescriptor.

`-incoming exec:`*`cmdline`*
: Accept incoming migration as an output from specified external command.

`-incoming defer`
: Wait for the URI to be specified via migrate_incoming. The monitor can be used to change settings (such as migration parameters) prior to issuing the migrate_incoming to allow the migration to begin.

`-nodefaults`
: Don't create default devices. Normally, QEMU sets the default devices like serial port, parallel port, virtual console, monitor device, VGA adapter, floppy and CD-ROM drive and others. The `-nodefaults` option will disable all those default devices.

`-chroot` *`dir`*
: Immediately before starting guest execution, chroot to the specified directory. Especially useful in combination with -runas.

`-runas` *`user`*
: Immediately before starting guest execution, drop root privileges, switching to the specified user.

`-prom-env` *`variable`*`=`*`value`*
: Set OpenBIOS nvram *variable* to given *value* (PPC, SPARC only).

`-semihosting`
: Enable semihosting mode (ARM, M68K, Xtensa, MIPS only).

`-semihosting-config`
`[enable=on|off][,target=native|gdb|auto][,arg=str[,...]]`
: Enable and configure semihosting (ARM, M68K, Xtensa, MIPS only).

  `target=native|gdb|auto`
  : Defines where the semihosting calls will be addressed, to QEMU (`native`) or to GDB (`gdb`). The default is `auto`, which means `gdb` during debug sessions and `native` otherwise.

  `arg=`*`str1`*`,arg=`*`str2`*`,...`
  : Allows the user to pass input arguments, and can be used multiple times to build up a list. The old-style `-kernel`/`-append` method of passing a command line is still supported for backward compatibility. If both the `--semihosting-config` `arg` and the `-kernel`/`-append` are specified, the former is passed to semihosting as it always takes precedence.

`-semihosting-cmdline`
: The `cmdline` defines the entire command line passed to the application via the semihosting calls, including the program name that will be passed as argv[0].

Must be the last option, all following arguments are passed to the application unchanged. (ARM, M68K, Xtensa only)

`-old-param`
Old param mode (ARM only).

`-sandbox` *arg*
Enable Seccomp mode 2 system call filter. 'on' will enable syscall filtering and 'off' will disable it. The default is 'off'.

`-readconfig` *file*
Read device configuration from *file*. This approach is useful when you want to spawn QEMU process with many command line options but you don't want to exceed the command line character limit.

`-writeconfig` *file*
Write device configuration to *file*. The *file* can be either filename to save command line and device configuration into file or dash `-`) character to print the output to stdout. This can be later used as input file for `-readconfig` option.

`-nodefconfig`
Normally QEMU loads configuration files from *sysconfdir* and *datadir* at startup. The `-nodefconfig` option will prevent QEMU from loading any of those config files.

`-no-user-config`
The `-no-user-config` option makes QEMU not load any of the user-provided config files on *sysconfdir*, but won't make it skip the QEMU-provided config files from *datadir*.

`-trace [[enable=]`*pattern*`][,events=`*file*`][,file=`*file*`]`
Specify tracing options.

`[enable=]`*pattern*
Immediately enable events matching *pattern*. The file must contain one event name (as listed in the `trace-events-all` file) per line; globbing patterns are accepted too. This option is only available if QEMU has been compiled with the *simple*, *stderr* or *ftrace* tracing backend. To specify multiple events or patterns, specify the `-trace` option multiple times.

Use `-trace help` to print a list of names of trace points.

`events=`*file*
Immediately enable events listed in *file*. The file must contain one event name (as listed in the `trace-events-all` file) per line; globbing patterns are accepted too. This option is only available if QEMU has been compiled with the *simple*, *stderr* or *ftrace* tracing backend.

`file=`*file*
Log output traces to *file*. This option is only available if QEMU has been compiled with the *simple* tracing backend.

`-enable-fips`
Enable FIPS 140-2 compliance mode.

`-msg timestamp[=on|off]`
prepend a timestamp to each log message.(default:on)

`-dump-vmstate` *file*
Dump json-encoded vmstate information for current machine type to file in *file*
Generic object creation

`-object` *typename*`[,`*prop1*`=`*value1*`,...]`
Create a new object of type *typename* setting properties in the order they are specified. Note that the 'id' property must be set. These objects are placed in the '/objects' path.

`-object memory-backend-file,id=`*id*`,size=`*size*`,mem-path=`*dir*`,share=`*on*`|`*off*
Creates a memory file backend object, which can be used to back the guest RAM with huge pages. The `id` parameter is a unique ID that will be used to reference this memory region when configuring the `-numa` argument. The `size` option provides the size of the memory region, and accepts common suffixes, eg `500M`. The `mem-path` provides the path to either a shared memory or huge page filesystem mount. The `share` boolean option determines whether the memory region is marked as private to QEMU, or shared. The latter allows a co-operating external process to access the QEMU memory region.

`-object rng-random,id=`*id*`,filename=`*/dev/random*
Creates a random number generator backend which obtains entropy from a device on the host. The `id` parameter is a unique ID that will be used to reference this entropy backend from the `virtio-rng` device. The `filename` parameter specifies which file to obtain entropy from and if omitted defaults to `/dev/random`.

`-object rng-egd,id=`*id*`,chardev=`*chardevid*
Creates a random number generator backend which obtains entropy from an external daemon running on the host. The `id` parameter is a unique ID that will be used to reference this entropy backend from the `virtio-rng` device. The `chardev` parameter is the unique ID of a character device backend that provides the connection to the RNG daemon.

`-object tls-creds-anon,id=`*id*`,endpoint=`*endpoint*`,dir=`*/path/to/cred/dir*`,verify-peer=`*on*`|`*off*
Creates a TLS anonymous credentials object, which can be used to provide TLS support on network backends. The `id` parameter is a unique ID which network backends will use to access the credentials. The `endpoint` is either `server` or `client` depending on whether the QEMU network backend that uses the credentials will be acting

as a client or as a server. If `verify-peer` is enabled (the default) then once the handshake is completed, the peer credentials will be verified, though this is a no-op for anonymous credentials.

The *dir* parameter tells QEMU where to find the credential files. For server endpoints, this directory may contain a file *dh-params.pem* providing diffie-hellman parameters to use for the TLS server. If the file is missing, QEMU will generate a set of DH parameters at startup. This is a computationally expensive operation that consumes random pool entropy, so it is recommended that a persistent set of parameters be generated upfront and saved.

`-object tls-creds-x509,id=`*id*`,endpoint=`*endpoint*`,dir=`*/path/to/cred/dir*`,verify-peer=`*on|off*`,passwordid=`*id*

Creates a TLS anonymous credentials object, which can be used to provide TLS support on network backends. The `id` parameter is a unique ID which network backends will use to access the credentials. The `endpoint` is either `server` or `client` depending on whether the QEMU network backend that uses the credentials will be acting as a client or as a server. If `verify-peer` is enabled (the default) then once the handshake is completed, the peer credentials will be verified. With x509 certificates, this implies that the clients must be provided with valid client certificates too.

The *dir* parameter tells QEMU where to find the credential files. For server endpoints, this directory may contain a file *dh-params.pem* providing diffie-hellman parameters to use for the TLS server. If the file is missing, QEMU will generate a set of DH parameters at startup. This is a computationally expensive operation that consumes random pool entropy, so it is recommended that a persistent set of parameters be generated upfront and saved.

For x509 certificate credentials the directory will contain further files providing the x509 certificates. The certificates must be stored in PEM format, in filenames *ca-cert.pem*, *ca-crl.pem* (optional), *server-cert.pem* (only servers), *server-key.pem* (only servers), *client-cert.pem* (only clients), and *client-key.pem* (only clients).

For the *server-key.pem* and *client-key.pem* files which contain sensitive private keys, it is possible to use an encrypted version by providing the *passwordid* parameter. This provides the ID of a previously created `secret` object containing the password for decryption.

`-object filter-buffer,id=`*id*`,netdev=`*netdevid*`,interval=`*t*`[,queue=`*all|rx|tx*`][,status=`*on|off*`]`

Interval *t* can't be 0, this filter batches the packet delivery: all packets arriving in a given interval on netdev *netdevid* are delayed

until the end of the interval. Interval is in microseconds. `status` is optional that indicate whether the netfilter is on (enabled) or off (disabled), the default status for netfilter will be 'on'.

queue *all|rx|tx* is an option that can be applied to any netfilter.

`all`: the filter is attached both to the receive and the transmit queue of the netdev (default).

`rx`: the filter is attached to the receive queue of the netdev, where it will receive packets sent to the netdev.

`tx`: the filter is attached to the transmit queue of the netdev, where it will receive packets sent by the netdev.

`-object filter-mirror,id=`*id*`,netdev=`*netdevid*`,outdev=`*chardevid*`[,queue=`*all|rx|tx*`]`
filter-mirror on netdev *netdevid*,mirror net packet to chardev *chardevid*

`-object filter-redirector,id=`*id*`,netdev=`*netdevid*`,indev=`*chardevid*`,` outdev=*chardevid*[,queue=*all|rx|tx*]

filter-redirector on netdev *netdevid*,redirect filter's net packet to chardev *chardevid*,and redirect indev's packet to filter. Create a filter-redirector we need to differ outdev id from indev id, id can not be the same. we can just use indev or outdev, but at least one of indev or outdev need to be specified.

`-object filter-rewriter,id=`*id*`,netdev=`*netdevid*`,rewriter-mode=`*mode*`[,queue=`*all|rx|tx*`]`
Filter-rewriter is a part of COLO project.It will rewrite tcp packet to secondary from primary to keep secondary tcp connection,and rewrite tcp packet to primary from secondary make tcp packet can be handled by client.

usage: colo secondary: -object filter-redirector,id=f1,netdev=hn0,queue=tx,indev=red
-object filter-redirector,id=f2,netdev=hn0,queue=rx,outdev=red1
-object filter-rewriter,id=rew0,netdev=hn0,queue=all

`-object filter-dump,id=`*id*`,netdev=`*dev*`[,file=`*filename*`][,maxlen=`*len*`]`
Dump the network traffic on netdev *dev* to the file specified by *filename*. At most *len* bytes (64k by default) per packet are stored. The file format is libpcap, so it can be analyzed with tools such as tcpdump or Wireshark.

`-object colo-compare,id=`*id*`,primary_in=`*chardevid*`,secondary_in=`*chardevid*`,` outdev=*chardevid*

Colo-compare gets packet from primary_in*chardevid* and secondary_in*chardevid*, than compare primary packet with secondary packet. If the packets are same, we will output primary packet to outdev*chardevid*, else we will notify colo-frame do checkpoint and send primary packet to outdev*chardevid*.

we must use it with the help of filter-mirror and filter-redirector.

```
primary:
-netdev tap,id=hn0,vhost=off,script=/etc/qemu-ifup,downscript=/etc/qemu
-device e1000,id=e0,netdev=hn0,mac=52:a4:00:12:78:66
-chardev socket,id=mirror0,host=3.3.3.3,port=9003,server,nowait
-chardev socket,id=compare1,host=3.3.3.3,port=9004,server,nowait
-chardev socket,id=compare0,host=3.3.3.3,port=9001,server,nowait
-chardev socket,id=compare0-0,host=3.3.3.3,port=9001
-chardev socket,id=compare_out,host=3.3.3.3,port=9005,server,nowait
-chardev socket,id=compare_out0,host=3.3.3.3,port=9005
-object filter-mirror,id=m0,netdev=hn0,queue=tx,outdev=mirror0
-object filter-redirector,netdev=hn0,id=redire0,queue=rx,indev=compare_
-object filter-redirector,netdev=hn0,id=redire1,queue=rx,outdev=compare
-object colo-compare,id=comp0,primary_in=compare0-0,secondary_in=compar

secondary:
-netdev tap,id=hn0,vhost=off,script=/etc/qemu-ifup,down script=/etc/qem
-device e1000,netdev=hn0,mac=52:a4:00:12:78:66
-chardev socket,id=red0,host=3.3.3.3,port=9003
-chardev socket,id=red1,host=3.3.3.3,port=9004
-object filter-redirector,id=f1,netdev=hn0,queue=tx,indev=red0
-object filter-redirector,id=f2,netdev=hn0,queue=rx,outdev=red1
```

If you want to know the detail of above command line, you can read the colo-compare git log.

`-object cryptodev-backend-builtin,id=`*id*`[,queues=`*queues*`]`

Creates a cryptodev backend which executes crypto opreation from the QEMU cipher APIS. The *id* parameter is a unique ID that will be used to reference this cryptodev backend from the `virtio-crypto` device. The *queues* parameter is optional, which specify the queue number of cryptodev backend, the default of *queues* is 1.

```
# qemu-system-x86_64 \
[...] \
-object cryptodev-backend-builtin,id=cryptodev0 \
-device virtio-crypto-pci,id=crypto0,cryptodev=cryptodev0 \
[...]
```

`-object`
`secret,id=`*id*`,data=`*string*`,format=`*raw*`|`*base64*`[,keyid=`*secretid*`,iv=`*string*`]`
`-object`
`secret,id=`*id*`,file=`*filename*`,format=`*raw*`|`*base64*`[,keyid=`*secretid*`,iv=`*string*`]`

Defines a secret to store a password, encryption key, or some other sensitive data. The sensitive data can either be passed directly via

the *data* parameter, or indirectly via the *file* parameter. Using the *data* parameter is insecure unless the sensitive data is encrypted.

The sensitive data can be provided in raw format (the default), or base64. When encoded as JSON, the raw format only supports valid UTF-8 characters, so base64 is recommended for sending binary data. QEMU will convert from which ever format is provided to the format it needs internally. eg, an RBD password can be provided in raw format, even though it will be base64 encoded when passed onto the RBD sever.

For added protection, it is possible to encrypt the data associated with a secret using the AES-256-CBC cipher. Use of encryption is indicated by providing the *keyid* and *iv* parameters. The *keyid* parameter provides the ID of a previously defined secret that contains the AES-256 decryption key. This key should be 32-bytes long and be base64 encoded. The *iv* parameter provides the random initialization vector used for encryption of this particular secret and should be a base64 encrypted string of the 16-byte IV.

The simplest (insecure) usage is to provide the secret inline

```
# $QEMU -object secret,id=sec0,data=letmein,format=raw
```

The simplest secure usage is to provide the secret via a file

echo -n "letmein" > mypasswd.txt # $QEMU -object secret,id=sec0,file=mypasswd.txt,format=raw

For greater security, AES-256-CBC should be used. To illustrate usage, consider the openssl command line tool which can encrypt the data. Note that when encrypting, the plaintext must be padded to the cipher block size (32 bytes) using the standard PKCS#5/6 compatible padding algorithm.

First a master key needs to be created in base64 encoding:

```
# openssl rand -base64 32 > key.b64
# KEY=$(base64 -d key.b64 | hexdump  -v -e '/1 "%02X"')
```

Each secret to be encrypted needs to have a random initialization vector generated. These do not need to be kept secret

```
# openssl rand -base64 16 > iv.b64
# IV=$(base64 -d iv.b64 | hexdump  -v -e '/1 "%02X"')
```

The secret to be defined can now be encrypted, in this case we're telling openssl to base64 encode the result, but it could be left as raw bytes if desired.

```
# SECRET=$(echo -n "letmein" |
openssl enc -aes-256-cbc -a -K $KEY -iv $IV)
```

When launching QEMU, create a master secret pointing to `key.b64` and specify that to be used to decrypt the user password. Pass the contents of `iv.b64` to the second secret

```
# $QEMU \
-object secret,id=secmaster0,format=base64,file=key.b64 \
-object secret,id=sec0,keyid=secmaster0,format=base64,\
data=$SECRET,iv=$(<iv.b64)
```

2.4 Keys in the graphical frontends

During the graphical emulation, you can use special key combinations to change modes. The default key mappings are shown below, but if you use `-alt-grab` then the modifier is Ctrl-Alt-Shift (instead of Ctrl-Alt) and if you use `-ctrl-grab` then the modifier is the right Ctrl key (instead of Ctrl-Alt):

`Ctrl-Alt-f`
: Toggle full screen

`Ctrl-Alt-+`
: Enlarge the screen

`Ctrl-Alt--`
: Shrink the screen

`Ctrl-Alt-u`
: Restore the screen's un-scaled dimensions

`Ctrl-Alt-n`
: Switch to virtual console 'n'. Standard console mappings are:

  1 Target system display

  2 Monitor

  3 Serial port

`Ctrl-Alt` Toggle mouse and keyboard grab.

In the virtual consoles, you can use `Ctrl-Up`, `Ctrl-Down`, `Ctrl-PageUp` and `Ctrl-PageDown` to move in the back log.

2.5 Keys in the character backend multiplexer

During emulation, if you are using a character backend multiplexer (which is the default if you are using `-nographic`) then several commands are available via an escape sequence. These key sequences all start with an escape character, which is `Ctrl-a` by default, but can be changed with `-echr`. The list below assumes you're using the default.

`Ctrl-a h` Print this help

`Ctrl-a x` Exit emulator

`Ctrl-a s` Save disk data back to file (if -snapshot)

`Ctrl-a t` Toggle console timestamps

`Ctrl-a b` Send break (magic sysrq in Linux)

`Ctrl-a c` Rotate between the frontends connected to the multiplexer (usually this switches between the monitor and the console)

`Ctrl-a Ctrl-a`
: Send the escape character to the frontend

2.6 QEMU Monitor

The QEMU monitor is used to give complex commands to the QEMU emulator. You can use it to:

- Remove or insert removable media images (such as CD-ROM or floppies).
- Freeze/unfreeze the Virtual Machine (VM) and save or restore its state from a disk file.
- Inspect the VM state without an external debugger.

2.6.1 Commands

The following commands are available:

`help or ? [`*cmd*`]`
: Show the help for all commands or just for command *cmd*.

`commit`
: Commit changes to the disk images (if -snapshot is used) or backing files. If the backing file is smaller than the snapshot, then the backing file will be resized to be the same size as the snapshot. If the snapshot is smaller than the backing file, the backing file will not be truncated. If you want the backing file to match the size of the smaller snapshot, you can safely truncate it yourself once the commit operation successfully completes.

`q or quit`
: Quit the emulator.

`block_resize`
: Resize a block image while a guest is running. Usually requires guest action to see the updated size. Resize to a lower size is supported, but should be used with extreme caution. Note that this command only resizes image files, it can not resize block devices like LVM volumes.

`block_stream`
: Copy data from a backing file into a block device.

`block_job_set_speed`
: Set maximum speed for a background block operation.

`block_job_cancel`
: Stop an active background block operation (streaming, mirroring).

`block_job_complete`
: Manually trigger completion of an active background block operation. For mirroring, this will switch the device to the destination path.

`block_job_pause`
: Pause an active block streaming operation.

`block_job_resume`
: Resume a paused block streaming operation.

`eject [-f]` *device*
: Eject a removable medium (use -f to force it).

`drive_del` *device*
: Remove host block device. The result is that guest generated IO is no longer submitted against the host device underlying the disk. Once a drive has been deleted, the QEMU Block layer returns -EIO which results in IO errors in the guest for applications that are reading/writing to the device. These errors are always reported to the guest, regardless of the drive's error actions (drive options rerror, werror).

`change` *device setting*
: Change the configuration of a device.

  `change` *diskdevice filename* [*format* [*read-only-mode*]]
  : Change the medium for a removable disk device to point to *filename*. eg

    ```
    (qemu) change ide1-cd0 /path/to/some.iso
    ```

    format is optional.

    read-only-mode may be used to change the read-only status of the device. It accepts the following values:

    retain Retains the current status; this is the default.

    read-only Makes the device read-only.

    read-write Makes the device writable.

  `change vnc` *display,options*
  : Change the configuration of the VNC server. The valid syntax for *display* and *options* are described at Section 2.3 [sec_invocation], page 3. eg

    ```
    (qemu) change vnc localhost:1
    ```

  `change vnc password` [*password*]
  : Change the password associated with the VNC server. If the new password is not supplied, the monitor will prompt for it to be entered. VNC passwords are only significant up to 8 letters. eg

    ```
    (qemu) change vnc password
    Password: ********
    ```

`screendump` *filename*
: Save screen into PPM image *filename*.

`logfile` *filename*
: Output logs to *filename*.

`trace-event`
: changes status of a trace event

`trace-file on|off|flush`
: Open, close, or flush the trace file. If no argument is given, the status of the trace file is displayed.

`log` *item1*`[,...]`
: Activate logging of the specified items.

`savevm [tag|id]`
: Create a snapshot of the whole virtual machine. If *tag* is provided, it is used as human readable identifier. If there is already a snapshot with the same tag or ID, it is replaced. More info at Section 2.7.3 [vm_snapshots], page 64.

`loadvm tag|id`
: Set the whole virtual machine to the snapshot identified by the tag *tag* or the unique snapshot ID *id*.

`delvm tag|id`
: Delete the snapshot identified by *tag* or *id*.

`singlestep [off]`
: Run the emulation in single step mode. If called with option off, the emulation returns to normal mode.

`stop`
: Stop emulation.

`c or cont`
: Resume emulation.

`system_wakeup`
: Wakeup guest from suspend.

`gdbserver [port]`
: Start gdbserver session (default *port*=1234)

`x/fmt addr`
: Virtual memory dump starting at *addr*.

`xp /fmt addr`
: Physical memory dump starting at *addr*.

  fmt is a format which tells the command how to format the data. Its syntax is: `/{count}{format}{size}`

  count
  : is the number of items to be dumped.

  format
  : can be x (hex), d (signed decimal), u (unsigned decimal), o (octal), c (char) or i (asm instruction).

  size
  : can be b (8 bits), h (16 bits), w (32 bits) or g (64 bits). On x86, `h` or `w` can be specified with the `i` format to respectively select 16 or 32 bit code instruction size.

  Examples:

  - Dump 10 instructions at the current instruction pointer:

    ```
    (qemu) x/10i $eip
    0x90107063:  ret
    0x90107064:  sti
    0x90107065:  lea    0x0(%esi,1),%esi
    0x90107069:  lea    0x0(%edi,1),%edi
    0x90107070:  ret
    0x90107071:  jmp    0x90107080
    0x90107073:  nop
    ```

```
0x90107074:  nop
0x90107075:  nop
0x90107076:  nop
```

- Dump 80 16 bit values at the start of the video memory.

```
(qemu) xp/80hx 0xb8000
0x000b8000: 0x0b50 0x0b6c 0x0b65 0x0b78 0x0b38 0x0b36 0x0b2f 0x0b42
0x000b8010: 0x0b6f 0x0b63 0x0b68 0x0b73 0x0b20 0x0b56 0x0b47 0x0b41
0x000b8020: 0x0b42 0x0b69 0x0b6f 0x0b73 0x0b20 0x0b63 0x0b75 0x0b72
0x000b8030: 0x0b72 0x0b65 0x0b6e 0x0b74 0x0b2d 0x0b63 0x0b76 0x0b73
0x000b8040: 0x0b20 0x0b30 0x0b35 0x0b20 0x0b4e 0x0b6f 0x0b76 0x0b20
0x000b8050: 0x0b32 0x0b30 0x0b30 0x0b33 0x0720 0x0720 0x0720 0x0720
0x000b8060: 0x0720 0x0720 0x0720 0x0720 0x0720 0x0720 0x0720 0x0720
0x000b8070: 0x0720 0x0720 0x0720 0x0720 0x0720 0x0720 0x0720 0x0720
0x000b8080: 0x0720 0x0720 0x0720 0x0720 0x0720 0x0720 0x0720 0x0720
0x000b8090: 0x0720 0x0720 0x0720 0x0720 0x0720 0x0720 0x0720 0x0720
```

`p` or `print/`*`fmt expr`*
: Print expression value. Only the *format* part of *fmt* is used.

`i/`*`fmt addr`* `[.`*`index`*`]`
: Read I/O port.

`o/`*`fmt addr val`*
: Write to I/O port.

`sendkey` *`keys`*
: Send *keys* to the guest. *keys* could be the name of the key or the raw value in hexadecimal format. Use `-` to press several keys simultaneously. Example:

  ```
  sendkey ctrl-alt-f1
  ```

  This command is useful to send keys that your graphical user interface intercepts at low level, such as `ctrl-alt-f1` in X Window.

`system_reset`
: Reset the system.

`system_powerdown`
: Power down the system (if supported).

`sum` *`addr size`*
: Compute the checksum of a memory region.

`usb_add` *`devname`*
: Add the USB device *devname*. For details of available devices see Section 2.11.1 [usb_devices], page 90,

`usb_del` *`devname`*
: Remove the USB device *devname* from the QEMU virtual USB hub. *devname* has the syntax `bus.addr`. Use the monitor command `info usb` to see the devices you can remove.

`device_add` *`config`*
: Add device.

`device_del` *`id`*
: Remove device *id*. *id* may be a short ID or a QOM object path.

`cpu index` Set the default CPU.

`mouse_move dx dy [dz]`
Move the active mouse to the specified coordinates *dx* *dy* with optional scroll axis *dz*.

`mouse_button val`
Change the active mouse button state *val* (1=L, 2=M, 4=R).

`mouse_set index`
Set which mouse device receives events at given *index*, index can be obtained with

`info mice`

`wavcapture filename [frequency [bits [channels]]]`
Capture audio into *filename*. Using sample rate *frequency* bits per sample *bits* and number of channels *channels*.

Defaults:

- Sample rate = 44100 Hz - CD quality
- Bits = 16
- Number of channels = 2 - Stereo

`stopcapture index`
Stop capture with a given *index*, index can be obtained with

`info capture`

`memsave addr size file`
save to disk virtual memory dump starting at *addr* of size *size*.

`pmemsave addr size file`
save to disk physical memory dump starting at *addr* of size *size*.

`boot_set bootdevicelist`
Define new values for the boot device list. Those values will override the values specified on the command line through the `-boot` option.

The values that can be specified here depend on the machine type, but are the same that can be specified in the `-boot` command line option.

`nmi cpu` Inject an NMI on the default CPU (x86/s390) or all CPUs (ppc64).

`ringbuf_write device data`
Write *data* to ring buffer character device *device*. *data* must be a UTF-8 string.

`ringbuf_read device`
Read and print up to *size* bytes from ring buffer character device *device*. Certain non-printable characters are printed \uXXXX, where XXXX is the character code in hexadecimal. Character \ is printed \\. Bug: can screw up when the buffer contains invalid UTF-8 sequences, NUL characters, after the ring buffer lost data, and when reading stops because the size limit is reached.

`migrate [-d] [-b] [-i] uri`
Migrate to *uri* (using -d to not wait for completion). -b for migration with full copy of disk -i for migration with incremental copy of disk (base image is shared)

`migrate_cancel`
: Cancel the current VM migration.

`migrate_incoming` *uri*
: Continue an incoming migration using the *uri* (that has the same syntax as the -incoming option).

`migrate_set_cache_size` *value*
: Set cache size to *value* (in bytes) for xbzrle migrations.

`migrate_set_speed` *value*
: Set maximum speed to *value* (in bytes) for migrations.

`migrate_set_downtime` *second*
: Set maximum tolerated downtime (in seconds) for migration.

`migrate_set_capability` *capability state*
: Enable/Disable the usage of a capability *capability* for migration.

`migrate_set_parameter` *parameter value*
: Set the parameter *parameter* for migration.

`migrate_start_postcopy`
: Switch in-progress migration to postcopy mode. Ignored after the end of migration (or once already in postcopy).

`x_colo_lost_heartbeat`
: Tell COLO that heartbeat is lost, a failover or takeover is needed.

`client_migrate_info` *protocol hostname port tls-port cert-subject*
: Set migration information for remote display. This makes the server ask the client to automatically reconnect using the new parameters once migration finished successfully. Only implemented for SPICE.

`dump-guest-memory [-p]` *filename begin length*
`dump-guest-memory [-z|-l|-s]` *filename*
: Dump guest memory to *protocol*. The file can be processed with crash or gdb. Without -z|-l|-s, the dump format is ELF. -p: do paging to get guest's memory mapping. -z: dump in kdump-compressed format, with zlib compression. -l: dump in kdump-compressed format, with lzo compression. -s: dump in kdump-compressed format, with snappy compression. filename: dump file name. begin: the starting physical address. It's optional, and should be specified together with length. length: the memory size, in bytes. It's optional, and should be specified together with begin.

`dump-skeys` *filename*
: Save guest storage keys to a file.

`snapshot_blkdev`
: Snapshot device, using snapshot file as target if provided

`snapshot_blkdev_internal`
: Take an internal snapshot on device if it support

`snapshot_delete_blkdev_internal`
: Delete an internal snapshot on device if it support

`drive_mirror`
: Start mirroring a block device's writes to a new destination, using the specified target.

`drive_backup`
: Start a point-in-time copy of a block device to a specificed target.

`drive_add`
: Add drive to PCI storage controller.

`pcie_aer_inject_error`
: Inject PCIe AER error

`host_net_add`
: Add host VLAN client.

`host_net_remove`
: Remove host VLAN client.

`netdev_add`
: Add host network device.

`netdev_del`
: Remove host network device.

`object_add`
: Create QOM object.

`object_del`
: Destroy QOM object.

`hostfwd_add`
: Redirect TCP or UDP connections from host to guest (requires -net user).

`hostfwd_remove`
: Remove host-to-guest TCP or UDP redirection.

`balloon` *`value`*
: Request VM to change its memory allocation to *value* (in MB).

`set_link` *`name`* `[on|off]`
: Switch link *name* on (i.e. up) or off (i.e. down).

`watchdog_action`
: Change watchdog action.

`acl_show` *`aclname`*
: List all the matching rules in the access control list, and the default policy. There are currently two named access control lists, *vnc.x509dname* and *vnc.username* matching on the x509 client certificate distinguished name, and SASL username respectively.

`acl_policy` *`aclname`* `allow|deny`
: Set the default access control list policy, used in the event that none of the explicit rules match. The default policy at startup is always `deny`.

`acl_add` *`aclname match`* `allow|deny [`*`index`*`]`
: Add a match rule to the access control list, allowing or denying access. The match will normally be an exact username or x509 distinguished name, but can optionally include wildcard globs. eg `*@EXAMPLE.COM` to allow all users in the `EXAMPLE.COM` kerberos realm. The match will normally be appended to the end of the ACL, but can be inserted earlier in the list if the optional *index* parameter is supplied.

`acl_remove` *`aclname match`*
: Remove the specified match rule from the access control list.

`acl_reset` *`aclname`*
: Remove all matches from the access control list, and set the default policy back to `deny`.

`nbd_server_start` *`host:port`*
: Start an NBD server on the given host and/or port. If the `-a` option is included, all of the virtual machine's block devices that have an inserted media on them are automatically exported; in this case, the `-w` option makes the devices writable too.

`nbd_server_add` *`device`*
: Export a block device through QEMU's NBD server, which must be started beforehand with `nbd_server_start`. The `-w` option makes the exported device writable too.

`nbd_server_stop`
: Stop the QEMU embedded NBD server.

`mce` *`cpu bank status mcgstatus addr misc`*
: Inject an MCE on the given CPU (x86 only).

`getfd` *`fdname`*
: If a file descriptor is passed alongside this command using the SCM_RIGHTS mechanism on unix sockets, it is stored using the name *fdname* for later use by other monitor commands.

`closefd` *`fdname`*
: Close the file descriptor previously assigned to *fdname* using the `getfd` command. This is only needed if the file descriptor was never used by another monitor command.

`block_passwd` *`device password`*
: Set the encrypted device *device* password to *password*

`block_set_io_throttle` *`device bps bps_rd bps_wr iops iops_rd iops_wr`*
: Change I/O throttle limits for a block drive to *bps bps_rd bps_wr iops iops_rd iops_wr*

`set_password [ vnc | spice ] password [ action-if-connected ]`
: Change spice/vnc password. Use zero to make the password stay valid forever. *action-if-connected* specifies what should happen in case a connection is established: *fail* makes the password change fail. *disconnect* changes the password

and disconnects the client. *keep* changes the password and keeps the connection up. *keep* is the default.

`expire_password [ vnc | spice ] expire-time`
Specify when a password for spice/vnc becomes invalid. *expire-time* accepts:

now Invalidate password instantly.

never Password stays valid forever.

+*nsec* Password stays valid for *nsec* seconds starting now.

nsec Password is invalidated at the given time. *nsec* are the seconds passed since 1970, i.e. unix epoch.

`chardev-add args`
chardev_add accepts the same parameters as the -chardev command line switch.

`chardev-remove id`
Removes the chardev *id*.

`qemu-io device command`
Executes a qemu-io command on the given block device.

`cpu-add id`
Add CPU with id *id*

`qom-list [path]`
Print QOM properties of object at location *path*

`qom-set path property value`
Set QOM property *property* of object at location *path* to value *value*

`info subcommand`
Show various information about the system state.

`info version`
Show the version of QEMU.

`info network`
Show the network state.

`info chardev`
Show the character devices.

`info block`
Show info of one block device or all block devices.

`info blockstats`
Show block device statistics.

`info block-jobs`
Show progress of ongoing block device operations.

`info registers`
Show the cpu registers.

`info lapic`
Show local APIC state

`info ioapic`
: Show io APIC state

`info cpus` Show infos for each CPU.

`info history`
: Show the command line history.

`info irq` Show the interrupts statistics (if available).

`info pic` Show i8259 (PIC) state.

`info pci` Show PCI information.

`info tlb` Show virtual to physical memory mappings.

`info mem` Show the active virtual memory mappings.

`info mtree`
: Show memory tree.

`info jit` Show dynamic compiler info.

`info opcount`
: Show dynamic compiler opcode counters

`info kvm` Show KVM information.

`info numa` Show NUMA information.

`info usb` Show guest USB devices.

`info usbhost`
: Show host USB devices.

`info profile`
: Show profiling information.

`info capture`
: Show capture information.

`info snapshots`
: Show the currently saved VM snapshots.

`info status`
: Show the current VM status (running|paused).

`info mice` Show which guest mouse is receiving events.

`info vnc` Show the vnc server status.

`info spice`
: Show the spice server status.

`info name` Show the current VM name.

`info uuid` Show the current VM UUID.

`info cpustats`
: Show CPU statistics.

`info usernet`
: Show user network stack connection states.

`info migrate`
: Show migration status.

`info migrate_capabilities`
: Show current migration capabilities.

`info migrate_parameters`
: Show current migration parameters.

`info migrate_cache_size`
: Show current migration xbzrle cache size.

`info balloon`
: Show balloon information.

`info qtree`
: Show device tree.

`info qdm` Show qdev device model list.

`info qom-tree`
: Show QOM composition tree.

`info roms` Show roms.

`info trace-events`
: Show available trace-events & their state.

`info tpm` Show the TPM device.

`info memdev`
: Show memory backends

`info memory-devices`
: Show memory devices.

`info iothreads`
: Show iothread's identifiers.

`info rocker` *`name`*
: Show rocker switch.

`info rocker_ports` *`name`*`-ports`
: Show rocker ports.

`info rocker_of_dpa_flows` *`name`* [*`tbl_id`*]
: Show rocker OF-DPA flow tables.

`info rocker-of-dpa-groups` *`name`* [*`type`*]
: Show rocker OF-DPA groups.

`info skeys` *`address`*
: Display the value of a storage key (s390 only)

`info dump` Display the latest dump status.

`info hotpluggable-cpus`
: Show information about hotpluggable CPUs

2.6.2 Integer expressions

The monitor understands integers expressions for every integer argument. You can use register names to get the value of specifics CPU registers by prefixing them with *$*.

2.7 Disk Images

Since version 0.6.1, QEMU supports many disk image formats, including growable disk images (their size increase as non empty sectors are written), compressed and encrypted disk images. Version 0.8.3 added the new qcow2 disk image format which is essential to support VM snapshots.

2.7.1 Quick start for disk image creation

You can create a disk image with the command:

```
qemu-img create myimage.img mysize
```

where *myimage.img* is the disk image filename and *mysize* is its size in kilobytes. You can add an `M` suffix to give the size in megabytes and a `G` suffix for gigabytes.

See Section 2.7.4 [qemu_img_invocation], page 65, for more information.

2.7.2 Snapshot mode

If you use the option `-snapshot`, all disk images are considered as read only. When sectors in written, they are written in a temporary file created in `/tmp`. You can however force the write back to the raw disk images by using the `commit` monitor command (or `C-a s` in the serial console).

2.7.3 VM snapshots

VM snapshots are snapshots of the complete virtual machine including CPU state, RAM, device state and the content of all the writable disks. In order to use VM snapshots, you must have at least one non removable and writable block device using the `qcow2` disk image format. Normally this device is the first virtual hard drive.

Use the monitor command `savevm` to create a new VM snapshot or replace an existing one. A human readable name can be assigned to each snapshot in addition to its numerical ID.

Use `loadvm` to restore a VM snapshot and `delvm` to remove a VM snapshot. `info snapshots` lists the available snapshots with their associated information:

```
(qemu) info snapshots
Snapshot devices: hda
Snapshot list (from hda):
ID        TAG                 VM SIZE                DATE       VM CLOCK
1         start                   41M 2006-08-06 12:38:02   00:00:14.954
2                                 40M 2006-08-06 12:43:29   00:00:18.633
3         msys                    40M 2006-08-06 12:44:04   00:00:23.514
```

A VM snapshot is made of a VM state info (its size is shown in `info snapshots`) and a snapshot of every writable disk image. The VM state info is stored in the first `qcow2` non removable and writable block device. The disk image snapshots are stored in every disk image. The size of a snapshot in a disk image is difficult to evaluate and is not shown by

`info snapshots` because the associated disk sectors are shared among all the snapshots to save disk space (otherwise each snapshot would need a full copy of all the disk images).

When using the (unrelated) `-snapshot` option (Section 2.7.2 [disk_images_snapshot_mode], page 64), you can always make VM snapshots, but they are deleted as soon as you exit QEMU.

VM snapshots currently have the following known limitations:

- They cannot cope with removable devices if they are removed or inserted after a snapshot is done.
- A few device drivers still have incomplete snapshot support so their state is not saved or restored properly (in particular USB).

2.7.4 `qemu-img` Invocation

`qemu-img [`*`standard options`*`] `*`command`*` [`*`command options`*`]`

qemu-img allows you to create, convert and modify images offline. It can handle all image formats supported by QEMU.

Warning: Never use qemu-img to modify images in use by a running virtual machine or any other process; this may destroy the image. Also, be aware that querying an image that is being modified by another process may encounter inconsistent state.

Standard options:

`-h, --help`
: Display this help and exit

`-V, --version`
: Display version information and exit

`-T, --trace [[enable=]`*`pattern`*`][,events=`*`file`*`][,file=`*`file`*`]`
: Specify tracing options.

  `[enable=]`*`pattern`*
  : Immediately enable events matching *pattern*. The file must contain one event name (as listed in the `trace-events-all` file) per line; globbing patterns are accepted too. This option is only available if QEMU has been compiled with the *simple*, *stderr* or *ftrace* tracing backend. To specify multiple events or patterns, specify the `-trace` option multiple times.

    Use `-trace help` to print a list of names of trace points.

  `events=`*`file`*
  : Immediately enable events listed in *file*. The file must contain one event name (as listed in the `trace-events-all` file) per line; globbing patterns are accepted too. This option is only available if QEMU has been compiled with the *simple*, *stderr* or *ftrace* tracing backend.

  `file=`*`file`*
  : Log output traces to *file*. This option is only available if QEMU has been compiled with the *simple* tracing backend.

The following commands are supported:

`bench [-c count] [-d depth] [-f fmt] [--flush-interval=flush_interval] [-n] [--no-drain] [-o offset] [--pattern=pattern] [-q] [-s buffer_size] [-S step_size] [-t cache] [-w] filename`
`check [--object objectdef] [--image-opts] [-q] [-f fmt] [--output=ofmt] [-r [leaks | all]] [-T src_cache] filename`
`create [--object objectdef] [--image-opts] [-q] [-f fmt] [-o options] filename [size]`
`commit [--object objectdef] [--image-opts] [-q] [-f fmt] [-t cache] [-b base] [-d] [-p] filename`
`compare [--object objectdef] [--image-opts] [-f fmt] [-F fmt] [-T src_cache] [-p] [-q] [-s] filename1 filename2`
`convert [--object objectdef] [--image-opts] [-c] [-p] [-q] [-n] [-f fmt] [-t cache] [-T src_cache] [-O output_fmt] [-o options] [-s snapshot_id_or_name] [-l snapshot_param] [-S sparse_size] filename [filename2 [...]] output_filename`
`dd [--image-opts] [-f fmt] [-O output_fmt] [bs=block_size] [count=blocks] [skip=blocks] if=input of=output`
`info [--object objectdef] [--image-opts] [-f fmt] [--output=ofmt] [--backing-chain] filename`
`map [--object objectdef] [--image-opts] [-f fmt] [--output=ofmt] filename`
`snapshot [--object objectdef] [--image-opts] [-q] [-l | -a snapshot | -c snapshot | -d snapshot] filename`
`rebase [--object objectdef] [--image-opts] [-q] [-f fmt] [-t cache] [-T src_cache] [-p] [-u] -b backing_file [-F backing_fmt] filename`
`resize [--object objectdef] [--image-opts] [-q] filename [+ | -]size`
`amend [--object objectdef] [--image-opts] [-p] [-q] [-f fmt] [-t cache] -o options filename`

Command parameters:

filename is a disk image filename

–object objectdef
is a QEMU user creatable object definition. See the `qemu(1)` manual page for a description of the object properties. The most common object type is a `secret`, which is used to supply passwords and/or encryption keys.

–image-opts
Indicates that the *filename* parameter is to be interpreted as a full option string, not a plain filename. This parameter is mutually exclusive with the *-f* and *-F* parameters.

fmt is the disk image format. It is guessed automatically in most cases. See below for a description of the supported disk formats.

–backing-chain
will enumerate information about backing files in a disk image chain. Refer below for further description.

size is the disk image size in bytes. Optional suffixes `k` or `K` (kilobyte, 1024) `M` (megabyte, 1024k) and `G` (gigabyte, 1024M) and T (terabyte, 1024G) are supported. `b` is ignored.

output_filename
is the destination disk image filename

output_fmt
is the destination format

options is a comma separated list of format specific options in a name=value format. Use `-o ?` for an overview of the options supported by the used format or see the format descriptions below for details.

snapshot_param
is param used for internal snapshot, format is 'snapshot.id=[ID],snapshot.name=[NAME]' or '[ID_OR_NAME]'

snapshot_id_or_name
is deprecated, use snapshot_param instead

-c indicates that target image must be compressed (qcow format only)

-h with or without a command shows help and lists the supported formats

-p display progress bar (compare, convert and rebase commands only). If the *-p* option is not used for a command that supports it, the progress is reported when the process receives a `SIGUSR1` signal.

-q Quiet mode - do not print any output (except errors). There's no progress bar in case both *-q* and *-p* options are used.

-S size indicates the consecutive number of bytes that must contain only zeros for qemu-img to create a sparse image during conversion. This value is rounded down to the nearest 512 bytes. You may use the common size suffixes like `k` for kilobytes.

-t cache specifies the cache mode that should be used with the (destination) file. See the documentation of the emulator's `-drive cache=...` option for allowed values.

-T src_cache
specifies the cache mode that should be used with the source file(s). See the documentation of the emulator's `-drive cache=...` option for allowed values.

Parameters to snapshot subcommand:

`snapshot` is the name of the snapshot to create, apply or delete

`-a` applies a snapshot (revert disk to saved state)

`-c` creates a snapshot

`-d` deletes a snapshot

`-l` lists all snapshots in the given image

Parameters to compare subcommand:

`-f` First image format

`-F` Second image format

`-s` Strict mode - fail on different image size or sector allocation

Parameters to convert subcommand:

`-n` Skip the creation of the target volume

Parameters to dd subcommand:

`bs=`*`block_size`*
defines the block size

`count=`*`blocks`*
sets the number of input blocks to copy

`if=`*`input`* sets the input file

`of=`*`output`*
sets the output file

`skip=`*`blocks`*
sets the number of input blocks to skip

Command description:

`bench [-c` *`count`*`] [-d` *`depth`*`] [-f` *`fmt`*`] [--flush-interval=`*`flush_interval`*`] [-n] [--no-drain] [-o` *`offset`*`] [--pattern=`*`pattern`*`] [-q] [-s` *`buffer_size`*`] [-S` *`step_size`*`] [-t` *`cache`*`] [-w]` *`filename`*
Run a simple sequential I/O benchmark on the specified image. If `-w` is specified, a write test is performed, otherwise a read test is performed.

A total number of *count* I/O requests is performed, each *buffer_size* bytes in size, and with *depth* requests in parallel. The first request starts at the position given by *offset*, each following request increases the current position by *step_size*. If *step_size* is not given, *buffer_size* is used for its value.

If *flush_interval* is specified for a write test, the request queue is drained and a flush is issued before new writes are made whenever the number of remaining requests is a multiple of *flush_interval*. If additionally `--no-drain` is specified, a flush is issued without draining the request queue first.

If `-n` is specified, the native AIO backend is used if possible. On Linux, this option only works if `-t none` or `-t directsync` is specified as well.

For write tests, by default a buffer filled with zeros is written. This can be overridden with a pattern byte specified by *pattern*.

`check [-f` *`fmt`*`] [--output=`*`ofmt`*`] [-r [leaks | all]] [-T` *`src_cache`*`]` *`filename`*
Perform a consistency check on the disk image *filename*. The command can output in the format *ofmt* which is either `human` or `json`.

If `-r` is specified, qemu-img tries to repair any inconsistencies found during the check. `-r leaks` repairs only cluster leaks, whereas `-r all` fixes all kinds of errors, with a higher risk of choosing the wrong fix or hiding corruption that has already occurred.

Only the formats `qcow2`, `qed` and `vdi` support consistency checks.

In case the image does not have any inconsistencies, check exits with `0`. Other exit codes indicate the kind of inconsistency found or if another error occurred. The following table summarizes all exit codes of the check subcommand:

`0` Check completed, the image is (now) consistent

`1` Check not completed because of internal errors

`2` Check completed, image is corrupted

`3` Check completed, image has leaked clusters, but is not corrupted

`63` Checks are not supported by the image format

If `-r` is specified, exit codes representing the image state refer to the state after (the attempt at) repairing it. That is, a successful `-r all` will yield the exit code 0, independently of the image state before.

`create [-f` *`fmt`*`] [-o` *`options`*`]` *`filename`* `[`*`size`*`]`

Create the new disk image *filename* of size *size* and format *fmt*. Depending on the file format, you can add one or more *options* that enable additional features of this format.

If the option *backing_file* is specified, then the image will record only the differences from *backing_file*. No size needs to be specified in this case. *backing_file* will never be modified unless you use the `commit` monitor command (or qemu-img commit).

The size can also be specified using the *size* option with `-o`, it doesn't need to be specified separately in this case.

`commit [-q] [-f` *`fmt`*`] [-t` *`cache`*`] [-b` *`base`*`] [-d] [-p]` *`filename`*

Commit the changes recorded in *filename* in its base image or backing file. If the backing file is smaller than the snapshot, then the backing file will be resized to be the same size as the snapshot. If the snapshot is smaller than the backing file, the backing file will not be truncated. If you want the backing file to match the size of the smaller snapshot, you can safely truncate it yourself once the commit operation successfully completes.

The image *filename* is emptied after the operation has succeeded. If you do not need *filename* afterwards and intend to drop it, you may skip emptying *filename* by specifying the `-d` flag.

If the backing chain of the given image file *filename* has more than one layer, the backing file into which the changes will be committed may be specified as *base* (which has to be part of *filename*'s backing chain). If *base* is not specified, the immediate backing file of the top image (which is *filename*) will be used. For reasons of consistency, explicitly specifying *base* will always imply `-d` (since emptying an image after committing to an indirect backing file would lead to different data being read from the image due to content in the intermediate backing chain overruling the commit target).

`compare [-f` *`fmt`*`] [-F` *`fmt`*`] [-T` *`src_cache`*`] [-p] [-s] [-q]` *`filename1 filename2`*

Check if two images have the same content. You can compare images with different format or settings.

The format is probed unless you specify it by *-f* (used for *filename1*) and/or *-F* (used for *filename2*) option.

By default, images with different size are considered identical if the larger image contains only unallocated and/or zeroed sectors in the area after the end of the other image. In addition, if any sector is not allocated in one image and contains only zero bytes in the second one, it is evaluated as equal. You can use Strict mode by specifying the *-s* option. When compare runs in Strict mode, it fails in case image size differs or a sector is allocated in one image and is not allocated in the second one.

By default, compare prints out a result message. This message displays information that both images are same or the position of the first different byte. In addition, result message can report different image size in case Strict mode is used.

Compare exits with `0` in case the images are equal and with `1` in case the images differ. Other exit codes mean an error occurred during execution and standard error output should contain an error message. The following table sumarizes all exit codes of the compare subcommand:

`0` Images are identical

`1` Images differ

`2` Error on opening an image

`3` Error on checking a sector allocation

`4` Error on reading data

`convert [-c] [-p] [-n] [-f` *`fmt`*`] [-t` *`cache`*`] [-T` *`src_cache`*`] [-O` *`output_fmt`*`] [-o` *`options`*`] [-s` *`snapshot_id_or_name`*`] [-l` *`snapshot_param`*`] [-S` *`sparse_size`*`] `*`filename`*` [`*`filename2`*` [...]] `*`output_filename`*

Convert the disk image *filename* or a snapshot *snapshot_param*(*snapshot_id_or_name* is deprecated) to disk image *output_filename* using format *output_fmt*. It can be optionally compressed (`-c` option) or use any format specific options like encryption (`-o` option).

Only the formats `qcow` and `qcow2` support compression. The compression is read-only. It means that if a compressed sector is rewritten, then it is rewritten as uncompressed data.

Image conversion is also useful to get smaller image when using a growable format such as `qcow`: the empty sectors are detected and suppressed from the destination image.

sparse_size indicates the consecutive number of bytes (defaults to 4k) that must contain only zeros for qemu-img to create a sparse image during conversion. If *sparse_size* is 0, the source will not be scanned for unallocated or zero sectors, and the destination image will always be fully allocated.

You can use the *backing_file* option to force the output image to be created as a copy on write image of the specified base image; the *backing_file* should have the same content as the input's base image, however the path, image format, etc may differ.

If the `-n` option is specified, the target volume creation will be skipped. This is useful for formats such as `rbd` if the target volume has already been created with site specific options that cannot be supplied through qemu-img.

`dd [-f` *fmt*`] [-O` *output_fmt*`] [bs=`*block_size*`] [count=`*blocks*`] [skip=`*blocks*`] if=`*input* `of=`*output*

Dd copies from *input* file to *output* file converting it from *fmt* format to *output_fmt* format.

The data is by default read and written using blocks of 512 bytes but can be modified by specifying *block_size*. If count=*blocks* is specified dd will stop reading input after reading *blocks* input blocks.

The size syntax is similar to dd(1)'s size syntax.

`info [-f` *fmt*`] [--output=`*ofmt*`] [--backing-chain]` *filename*

Give information about the disk image *filename*. Use it in particular to know the size reserved on disk which can be different from the displayed size. If VM snapshots are stored in the disk image, they are displayed too. The command can output in the format *ofmt* which is either `human` or `json`.

If a disk image has a backing file chain, information about each disk image in the chain can be recursively enumerated by using the option `--backing-chain`.

For instance, if you have an image chain like:

```
base.qcow2 <- snap1.qcow2 <- snap2.qcow2
```

To enumerate information about each disk image in the above chain, starting from top to base, do:

```
qemu-img info --backing-chain snap2.qcow2
```

`map [-f` *fmt*`] [--output=`*ofmt*`]` *filename*

Dump the metadata of image *filename* and its backing file chain. In particular, this commands dumps the allocation state of every sector of *filename*, together with the topmost file that allocates it in the backing file chain.

Two option formats are possible. The default format (`human`) only dumps known-nonzero areas of the file. Known-zero parts of the file are omitted altogether, and likewise for parts that are not allocated throughout the chain. `qemu-img` output will identify a file from where the data can be read, and the offset in the file. Each line will include four fields, the first three of which are hexadecimal numbers. For example the first line of:

```
Offset          Length          Mapped to       File
0               0x20000         0x50000         /tmp/overlay.qcow2
0x100000        0x10000         0x95380000      /tmp/backing.qcow2
```

means that 0x20000 (131072) bytes starting at offset 0 in the image are available in /tmp/overlay.qcow2 (opened in `raw` format) starting at offset 0x50000 (327680). Data that is compressed, encrypted, or otherwise not available in raw format will cause an error if `human` format is in use. Note that file names can include newlines, thus it is not safe to parse this output format in scripts.

The alternative format `json` will return an array of dictionaries in JSON format. It will include similar information in the `start`, `length`, `offset` fields; it will also include other more specific information:

- whether the sectors contain actual data or not (boolean field `data`; if false, the sectors are either unallocated or stored as optimized all-zero clusters);
- whether the data is known to read as zero (boolean field `zero`);
- in order to make the output shorter, the target file is expressed as a `depth`; for example, a depth of 2 refers to the backing file of the backing file of *filename*.

In JSON format, the `offset` field is optional; it is absent in cases where `human` format would omit the entry or exit with an error. If `data` is false and the `offset` field is present, the corresponding sectors in the file are not yet in use, but they are preallocated.

For more information, consult `include/block/block.h` in QEMU's source code.

`snapshot [-l | -a` *snapshot* `| -c` *snapshot* `| -d` *snapshot* `]` *filename*

List, apply, create or delete snapshots in image *filename*.

`rebase [-f` *fmt*`] [-t` *cache*`] [-T` *src_cache*`] [-p] [-u] -b` *backing_file* `[-F` *backing_fmt*`]` *filename*

Changes the backing file of an image. Only the formats `qcow2` and `qed` support changing the backing file.

The backing file is changed to *backing_file* and (if the image format of *filename* supports this) the backing file format is changed to *backing_fmt*. If *backing_file* is specified as "" (the empty string), then the image is rebased onto no backing file (i.e. it will exist independently of any backing file).

cache specifies the cache mode to be used for *filename*, whereas *src_cache* specifies the cache mode for reading backing files.

There are two different modes in which `rebase` can operate:

`Safe mode` This is the default mode and performs a real rebase operation. The new backing file may differ from the old one and qemu-img rebase will take care of keeping the guest-visible content of *filename* unchanged.

In order to achieve this, any clusters that differ between *backing_file* and the old backing file of *filename* are merged into *filename* before actually changing the backing file.

Note that the safe mode is an expensive operation, comparable to converting an image. It only works if the old backing file still exists.

`Unsafe mode`

qemu-img uses the unsafe mode if `-u` is specified. In this mode, only the backing file name and format of *filename* is changed without any checks on the file contents. The user must take care of specifying the correct new backing file, or the guest-visible content of the image will be corrupted.

This mode is useful for renaming or moving the backing file to somewhere else. It can be used without an accessible old backing

file, i.e. you can use it to fix an image whose backing file has already been moved/renamed.

You can use `rebase` to perform a "diff" operation on two disk images. This can be useful when you have copied or cloned a guest, and you want to get back to a thin image on top of a template or base image.

Say that `base.img` has been cloned as `modified.img` by copying it, and that the `modified.img` guest has run so there are now some changes compared to `base.img`. To construct a thin image called `diff.qcow2` that contains just the differences, do:

```
qemu-img create -f qcow2 -b modified.img diff.qcow2
qemu-img rebase -b base.img diff.qcow2
```

At this point, `modified.img` can be discarded, since `base.img` + `diff.qcow2` contains the same information.

`resize` *`filename`* `[+ | -]`*`size`*

Change the disk image as if it had been created with *size*.

Before using this command to shrink a disk image, you MUST use file system and partitioning tools inside the VM to reduce allocated file systems and partition sizes accordingly. Failure to do so will result in data loss!

After using this command to grow a disk image, you must use file system and partitioning tools inside the VM to actually begin using the new space on the device.

`amend [-p] [-f` *`fmt`*`] [-t` *`cache`*`] -o` *`options filename`*

Amends the image format specific *options* for the image file *filename*. Not all file formats support this operation.

2.7.5 `qemu-nbd` Invocation

`qemu-nbd [OPTION]...` *`filename`*

`qemu-nbd -d` *`dev`*

Export a QEMU disk image using the NBD protocol.

filename is a disk image filename, or a set of block driver options if *–image-opts* is specified.

dev is an NBD device.

`--object type,id=`*`id`*`,...props...`

Define a new instance of the *type* object class identified by *id*. See the `qemu(1)` manual page for full details of the properties supported. The common object types that it makes sense to define are the `secret` object, which is used to supply passwords and/or encryption keys, and the `tls-creds` object, which is used to supply TLS credentials for the qemu-nbd server.

`-p, --port=`*`port`*

The TCP port to listen on (default '`10809`')

`-o, --offset=`*`offset`*

The offset into the image

`-b, --bind=`*iface*
: The interface to bind to (default '`0.0.0.0`')

`-k, --socket=`*path*
: Use a unix socket with path *path*

`--image-opts`
: Treat *filename* as a set of image options, instead of a plain filename. If this flag is specified, the *-f* flag should not be used, instead the '`format=`' option should be set.

`-f, --format=`*fmt*
: Force the use of the block driver for format *fmt* instead of auto-detecting

`-r, --read-only`
: Export the disk as read-only

`-P, --partition=`*num*
: Only expose partition *num*

`-s, --snapshot`
: Use *filename* as an external snapshot, create a temporary file with backing_file=*filename*, redirect the write to the temporary one

`-l, --load-snapshot=`*snapshot_param*
: Load an internal snapshot inside *filename* and export it as an read-only device, *snapshot_param* format is 'snapshot.id=[ID],snapshot.name=[NAME]' or '[ID_OR_NAME]'

`-n, --nocache`
`--cache=`*cache*
: The cache mode to be used with the file. See the documentation of the emulator's `-drive cache=...` option for allowed values.

`--aio=`*aio*
: Set the asynchronous I/O mode between '`threads`' (the default) and '`native`' (Linux only).

`--discard=`*discard*
: Control whether *discard* (also known as *trim* or *unmap*) requests are ignored or passed to the filesystem. *discard* is one of '`ignore`' (or '`off`'), '`unmap`' (or '`on`'). The default is '`ignore`'.

`--detect-zeroes=`*detect-zeroes*
: Control the automatic conversion of plain zero writes by the OS to driver-specific optimized zero write commands. *detect-zeroes* is one of '`off`', '`on`' or '`unmap`'. '`unmap`' converts a zero write to an unmap operation and can only be used if *discard* is set to '`unmap`'. The default is '`off`'.

`-c, --connect=`*dev*
: Connect *filename* to NBD device *dev*

`-d, --disconnect`
: Disconnect the device *dev*

`-e, --shared=`*num*
: Allow up to *num* clients to share the device (default '`1`')

`-t, --persistent`
: Don't exit on the last connection

`-x, --export-name=`*name*
: Set the NBD volume export name. This switches the server to use the new style NBD protocol negotiation

`-D, --description=`*description*
: Set the NBD volume export description, as a human-readable string. Requires the use of `-x`

`--tls-creds=ID`
: Enable mandatory TLS encryption for the server by setting the ID of the TLS credentials object previously created with the –object option.

`--fork` Fork off the server process and exit the parent once the server is running.

`-v, --verbose`
: Display extra debugging information

`-h, --help`
: Display this help and exit

`-V, --version`
: Display version information and exit

`-T, --trace [[enable=]`*pattern*`][,events=`*file*`][,file=`*file*`]`
: Specify tracing options.

  `[enable=]`*pattern*
  : Immediately enable events matching *pattern*. The file must contain one event name (as listed in the `trace-events-all` file) per line; globbing patterns are accepted too. This option is only available if QEMU has been compiled with the *simple*, *stderr* or *ftrace* tracing backend. To specify multiple events or patterns, specify the `-trace` option multiple times.

    Use `-trace help` to print a list of names of trace points.

  `events=`*file*
  : Immediately enable events listed in *file*. The file must contain one event name (as listed in the `trace-events-all` file) per line; globbing patterns are accepted too. This option is only available if QEMU has been compiled with the *simple*, *stderr* or *ftrace* tracing backend.

  `file=`*file*
  : Log output traces to *file*. This option is only available if QEMU has been compiled with the *simple* tracing backend.

2.7.6 qemu-ga Invocation

`qemu-ga [OPTIONS]`

The QEMU Guest Agent is a daemon intended to be run within virtual machines. It allows the hypervisor host to perform various operations in the guest, such as:

- get information from the guest
- set the guest's system time
- read/write a file
- sync and freeze the filesystems
- suspend the guest
- reconfigure guest local processors
- set user's password
- ...

qemu-ga will read a system configuration file on startup (located at `/etc/qemu/qemu-ga.conf` by default), then parse remaining configuration options on the command line. For the same key, the last option wins, but the lists accumulate (see below for configuration file format).

`-m, --method=method`
: Transport method: one of '`unix-listen`', '`virtio-serial`', or '`isa-serial`' ('`virtio-serial`' is the default).

`-p, --path=path`
: Device/socket path (the default for virtio-serial is '`/dev/virtio-ports/org.qemu.guest_agent.0`' the default for isa-serial is '`/dev/ttyS0`')

`-l, --logfile=path`
: Set log file path (default is stderr).

`-f, --pidfile=path`
: Specify pid file (default is '`/var/run/qemu-ga.pid`').

`-F, --fsfreeze-hook=path`
: Enable fsfreeze hook. Accepts an optional argument that specifies script to run on freeze/thaw. Script will be called with 'freeze'/'thaw' arguments accordingly (default is '`/etc/qemu/fsfreeze-hook`'). If using -F with an argument, do not follow -F with a space (for example: '`-F/var/run/fsfreezehook.sh`').

`-t, --statedir=path`
: Specify the directory to store state information (absolute paths only, default is '`/var/run`').

`-v, --verbose`
: Log extra debugging information.

`-V, --version`
: Print version information and exit.

`-d, --daemon`
: Daemonize after startup (detach from terminal).

`-b, --blacklist=`*list*
: Comma-separated list of RPCs to disable (no spaces, '`?`' to list available RPCs).

`-D, --dump-conf`
: Dump the configuration in a format compatible with `qemu-ga.conf` and exit.

`-h, --help`
: Display this help and exit.

The syntax of the `qemu-ga.conf` configuration file follows the Desktop Entry Specification, here is a quick summary: it consists of groups of key-value pairs, interspersed with comments.

```
# qemu-ga configuration sample
[general]
daemonize = 0
pidfile = /var/run/qemu-ga.pid
verbose = 0
method = virtio-serial
path = /dev/virtio-ports/org.qemu.guest_agent.0
statedir = /var/run
```

The list of keys follows the command line options:

```
daemon= boolean
method= string
path= string
logfile= string
pidfile= string
fsfreeze-hook= string
statedir= string
verbose= boolean
blacklist= string list
```

2.7.7 Disk image file formats

QEMU supports many image file formats that can be used with VMs as well as with any of the tools (like `qemu-img`). This includes the preferred formats raw and qcow2 as well as formats that are supported for compatibility with older QEMU versions or other hypervisors.

Depending on the image format, different options can be passed to `qemu-img create` and `qemu-img convert` using the `-o` option. This section describes each format and the options that are supported for it.

`raw`
: Raw disk image format. This format has the advantage of being simple and easily exportable to all other emulators. If your file system supports *holes* (for example in ext2 or ext3 on Linux or NTFS on Windows), then only the written sectors will reserve space. Use `qemu-img info` to know the real size used by the image or `ls -ls` on Unix/Linux.

  Supported options:

`preallocation`
: Preallocation mode (allowed values: `off`, `falloc`, `full`). `falloc` mode preallocates space for image by calling posix_fallocate(). `full` mode preallocates space for image by writing zeros to underlying storage.

`qcow2`
: QEMU image format, the most versatile format. Use it to have smaller images (useful if your filesystem does not supports holes, for example on Windows), zlib based compression and support of multiple VM snapshots.

  Supported options:

  `compat`
  : Determines the qcow2 version to use. `compat=0.10` uses the traditional image format that can be read by any QEMU since 0.10. `compat=1.1` enables image format extensions that only QEMU 1.1 and newer understand (this is the default). Amongst others, this includes zero clusters, which allow efficient copy-on-read for sparse images.

  `backing_file`
  : File name of a base image (see `create` subcommand)

  `backing_fmt`
  : Image format of the base image

  `encryption`
  : If this option is set to `on`, the image is encrypted with 128-bit AES-CBC.

    The use of encryption in qcow and qcow2 images is considered to be flawed by modern cryptography standards, suffering from a number of design problems:

    - The AES-CBC cipher is used with predictable initialization vectors based on the sector number. This makes it vulnerable to chosen plaintext attacks which can reveal the existence of encrypted data.
    - The user passphrase is directly used as the encryption key. A poorly chosen or short passphrase will compromise the security of the encryption.
    - In the event of the passphrase being compromised there is no way to change the passphrase to protect data in any qcow images. The files must be cloned, using a different encryption passphrase in the new file. The original file must then be securely erased using a program like shred, though even this is ineffective with many modern storage technologies.

    Use of qcow / qcow2 encryption with QEMU is deprecated, and support for it will go away in a future release. Users are recommended to use an alternative encryption technology such as the Linux dm-crypt / LUKS system.

`cluster_size`
: Changes the qcow2 cluster size (must be between 512 and 2M). Smaller cluster sizes can improve the image file size whereas larger cluster sizes generally provide better performance.

`preallocation`
: Preallocation mode (allowed values: `off`, `metadata`, `falloc`, `full`). An image with preallocated metadata is initially larger but can improve performance when the image needs to grow. `falloc` and `full` preallocations are like the same options of `raw` format, but sets up metadata also.

`lazy_refcounts`
: If this option is set to `on`, reference count updates are postponed with the goal of avoiding metadata I/O and improving performance. This is particularly interesting with `cache=writethrough` which doesn't batch metadata updates. The tradeoff is that after a host crash, the reference count tables must be rebuilt, i.e. on the next open an (automatic) `qemu-img check -r all` is required, which may take some time.

  This option can only be enabled if `compat=1.1` is specified.

`nocow`
: If this option is set to `on`, it will turn off COW of the file. It's only valid on btrfs, no effect on other file systems.

  Btrfs has low performance when hosting a VM image file, even more when the guest on the VM also using btrfs as file system. Turning off COW is a way to mitigate this bad performance. Generally there are two ways to turn off COW on btrfs: a) Disable it by mounting with nodatacow, then all newly created files will be NOCOW. b) For an empty file, add the NOCOW file attribute. That's what this option does.

  Note: this option is only valid to new or empty files. If there is an existing file which is COW and has data blocks already, it couldn't be changed to NOCOW by setting `nocow=on`. One can issue `lsattr filename` to check if the NOCOW flag is set or not (Capital 'C' is NOCOW flag).

`qed`
: Old QEMU image format with support for backing files and compact image files (when your filesystem or transport medium does not support holes).

  When converting QED images to qcow2, you might want to consider using the `lazy_refcounts=on` option to get a more QED-like behaviour.

  Supported options:

  `backing_file`
  : File name of a base image (see `create` subcommand).

  `backing_fmt`
  : Image file format of backing file (optional). Useful if the format cannot be autodetected because it has no header, like some vhd/vpc files.

`cluster_size`
: Changes the cluster size (must be power-of-2 between 4K and 64K). Smaller cluster sizes can improve the image file size whereas larger cluster sizes generally provide better performance.

`table_size`
: Changes the number of clusters per L1/L2 table (must be power-of-2 between 1 and 16). There is normally no need to change this value but this option can be used for performance benchmarking.

`qcow` Old QEMU image format with support for backing files, compact image files, encryption and compression.

Supported options:

`backing_file`
: File name of a base image (see `create` subcommand)

`encryption`
: If this option is set to `on`, the image is encrypted.

`vdi` VirtualBox 1.1 compatible image format. Supported options:

`static` If this option is set to `on`, the image is created with metadata preallocation.

`vmdk` VMware 3 and 4 compatible image format.

Supported options:

`backing_file`
: File name of a base image (see `create` subcommand).

`compat6` Create a VMDK version 6 image (instead of version 4)

`hwversion`
: Specify vmdk virtual hardware version. Compat6 flag cannot be enabled if hwversion is specified.

`subformat`
: Specifies which VMDK subformat to use. Valid options are `monolithicSparse` (default), `monolithicFlat`, `twoGbMaxExtentSparse`, `twoGbMaxExtentFlat` and `streamOptimized`.

`vpc` VirtualPC compatible image format (VHD). Supported options:

`subformat`
: Specifies which VHD subformat to use. Valid options are `dynamic` (default) and `fixed`.

`VHDX` Hyper-V compatible image format (VHDX). Supported options:

`subformat`
: Specifies which VHDX subformat to use. Valid options are `dynamic` (default) and `fixed`.

`block_state_zero`
: Force use of payload blocks of type 'ZERO'. Can be set to `on` (default) or `off`. When set to `off`, new blocks will be created as `PAYLOAD_BLOCK_NOT_PRESENT`, which means parsers are free to return arbitrary data for those blocks. Do not set to `off` when using `qemu-img convert` with `subformat=dynamic`.

`block_size`
: Block size; min 1 MB, max 256 MB. 0 means auto-calculate based on image size.

`log_size` Log size; min 1 MB.

2.7.7.1 Read-only formats

More disk image file formats are supported in a read-only mode.

`bochs` Bochs images of `growing` type.

`cloop` Linux Compressed Loop image, useful only to reuse directly compressed CD-ROM images present for example in the Knoppix CD-ROMs.

`dmg` Apple disk image.

`parallels`
: Parallels disk image format.

2.7.8 Using host drives

In addition to disk image files, QEMU can directly access host devices. We describe here the usage for QEMU version >= 0.8.3.

2.7.8.1 Linux

On Linux, you can directly use the host device filename instead of a disk image filename provided you have enough privileges to access it. For example, use `/dev/cdrom` to access to the CDROM.

`CD` You can specify a CDROM device even if no CDROM is loaded. QEMU has specific code to detect CDROM insertion or removal. CDROM ejection by the guest OS is supported. Currently only data CDs are supported.

`Floppy` You can specify a floppy device even if no floppy is loaded. Floppy removal is currently not detected accurately (if you change floppy without doing floppy access while the floppy is not loaded, the guest OS will think that the same floppy is loaded). Use of the host's floppy device is deprecated, and support for it will be removed in a future release.

`Hard disks`
: Hard disks can be used. Normally you must specify the whole disk (`/dev/hdb` instead of `/dev/hdb1`) so that the guest OS can see it as a partitioned disk. WARNING: unless you know what you do, it is better to only make READ-ONLY accesses to the hard disk otherwise you may corrupt your host data (use the `-snapshot` command line option or modify the device permissions accordingly).

2.7.8.2 Windows

`CD`
: The preferred syntax is the drive letter (e.g. `d:`). The alternate syntax `\\.\d:` is supported. `/dev/cdrom` is supported as an alias to the first CDROM drive.

  Currently there is no specific code to handle removable media, so it is better to use the `change` or `eject` monitor commands to change or eject media.

`Hard disks`
: Hard disks can be used with the syntax: `\\.\PhysicalDrive`*N* where *N* is the drive number (0 is the first hard disk).

  WARNING: unless you know what you do, it is better to only make READ-ONLY accesses to the hard disk otherwise you may corrupt your host data (use the `-snapshot` command line so that the modifications are written in a temporary file).

2.7.8.3 Mac OS X

`/dev/cdrom` is an alias to the first CDROM.

Currently there is no specific code to handle removable media, so it is better to use the `change` or `eject` monitor commands to change or eject media.

2.7.9 Virtual FAT disk images

QEMU can automatically create a virtual FAT disk image from a directory tree. In order to use it, just type:

```
qemu-system-i386 linux.img -hdb fat:/my_directory
```

Then you access access to all the files in the `/my_directory` directory without having to copy them in a disk image or to export them via SAMBA or NFS. The default access is *read-only*.

Floppies can be emulated with the `:floppy:` option:

```
qemu-system-i386 linux.img -fda fat:floppy:/my_directory
```

A read/write support is available for testing (beta stage) with the `:rw:` option:

```
qemu-system-i386 linux.img -fda fat:floppy:rw:/my_directory
```

What you should *never* do:

- use non-ASCII filenames ;
- use "-snapshot" together with ":rw:" ;
- expect it to work when loadvm'ing ;
- write to the FAT directory on the host system while accessing it with the guest system.

2.7.10 NBD access

QEMU can access directly to block device exported using the Network Block Device protocol.

```
qemu-system-i386 linux.img -hdb nbd://my_nbd_server.mydomain.org:1024/
```

If the NBD server is located on the same host, you can use an unix socket instead of an inet socket:

```
qemu-system-i386 linux.img -hdb nbd+unix://?socket=/tmp/my_socket
```

In this case, the block device must be exported using qemu-nbd:

```
qemu-nbd --socket=/tmp/my_socket my_disk.qcow2
```

The use of qemu-nbd allows sharing of a disk between several guests:

```
qemu-nbd --socket=/tmp/my_socket --share=2 my_disk.qcow2
```

and then you can use it with two guests:

```
qemu-system-i386 linux1.img -hdb nbd+unix://?socket=/tmp/my_socket
qemu-system-i386 linux2.img -hdb nbd+unix://?socket=/tmp/my_socket
```

If the nbd-server uses named exports (supported since NBD 2.9.18, or with QEMU's own embedded NBD server), you must specify an export name in the URI:

```
qemu-system-i386 -cdrom nbd://localhost/debian-500-ppc-netinst
qemu-system-i386 -cdrom nbd://localhost/openSUSE-11.1-ppc-netinst
```

The URI syntax for NBD is supported since QEMU 1.3. An alternative syntax is also available. Here are some example of the older syntax:

```
qemu-system-i386 linux.img -hdb nbd:my_nbd_server.mydomain.org:1024
qemu-system-i386 linux2.img -hdb nbd:unix:/tmp/my_socket
qemu-system-i386 -cdrom nbd:localhost:10809:exportname=debian-500-ppc-netinst
```

2.7.11 Sheepdog disk images

Sheepdog is a distributed storage system for QEMU. It provides highly available block level storage volumes that can be attached to QEMU-based virtual machines.

You can create a Sheepdog disk image with the command:

```
qemu-img create sheepdog:///image size
```

where *image* is the Sheepdog image name and *size* is its size.

To import the existing *filename* to Sheepdog, you can use a convert command.

```
qemu-img convert filename sheepdog:///image
```

You can boot from the Sheepdog disk image with the command:

```
qemu-system-i386 sheepdog:///image
```

You can also create a snapshot of the Sheepdog image like qcow2.

```
qemu-img snapshot -c tag sheepdog:///image
```

where *tag* is a tag name of the newly created snapshot.

To boot from the Sheepdog snapshot, specify the tag name of the snapshot.

```
qemu-system-i386 sheepdog:///image#tag
```

You can create a cloned image from the existing snapshot.

```
qemu-img create -b sheepdog:///base#tag sheepdog:///image
```

where *base* is a image name of the source snapshot and *tag* is its tag name.

You can use an unix socket instead of an inet socket:

```
qemu-system-i386 sheepdog+unix:///image?socket=path
```

If the Sheepdog daemon doesn't run on the local host, you need to specify one of the Sheepdog servers to connect to.

```
qemu-img create sheepdog://hostname:port/image size
qemu-system-i386 sheepdog://hostname:port/image
```

2.7.12 iSCSI LUNs

iSCSI is a popular protocol used to access SCSI devices across a computer network.

There are two different ways iSCSI devices can be used by QEMU.

The first method is to mount the iSCSI LUN on the host, and make it appear as any other ordinary SCSI device on the host and then to access this device as a /dev/sd device from QEMU. How to do this differs between host OSes.

The second method involves using the iSCSI initiator that is built into QEMU. This provides a mechanism that works the same way regardless of which host OS you are running QEMU on. This section will describe this second method of using iSCSI together with QEMU.

In QEMU, iSCSI devices are described using special iSCSI URLs

```
URL syntax:
iscsi://[<username>[%<password>]@]<host>[:<port>]/<target-iqn-name>/<lun>
```

Username and password are optional and only used if your target is set up using CHAP authentication for access control. Alternatively the username and password can also be set via environment variables to have these not show up in the process list

```
export LIBISCSI_CHAP_USERNAME=<username>
export LIBISCSI_CHAP_PASSWORD=<password>
iscsi://<host>/<target-iqn-name>/<lun>
```

Various session related parameters can be set via special options, either in a configuration file provided via '-readconfig' or directly on the command line.

If the initiator-name is not specified qemu will use a default name of 'iqn.2008-11.org.linux-kvm[:<name>'] where <name> is the name of the virtual machine.

```
Setting a specific initiator name to use when logging in to the target
-iscsi initiator-name=iqn.qemu.test:my-initiator

Controlling which type of header digest to negotiate with the target
-iscsi header-digest=CRC32C|CRC32C-NONE|NONE-CRC32C|NONE
```

These can also be set via a configuration file

```
[iscsi]
  user = "CHAP username"
  password = "CHAP password"
  initiator-name = "iqn.qemu.test:my-initiator"
  # header digest is one of CRC32C|CRC32C-NONE|NONE-CRC32C|NONE
  header-digest = "CRC32C"
```

Setting the target name allows different options for different targets

```
[iscsi "iqn.target.name"]
  user = "CHAP username"
  password = "CHAP password"
  initiator-name = "iqn.qemu.test:my-initiator"
  # header digest is one of CRC32C|CRC32C-NONE|NONE-CRC32C|NONE
  header-digest = "CRC32C"
```

Howto use a configuration file to set iSCSI configuration options:

```
cat >iscsi.conf <<EOF
[iscsi]
```

```
  user = "me"
  password = "my password"
  initiator-name = "iqn.qemu.test:my-initiator"
  header-digest = "CRC32C"
EOF

qemu-system-i386 -drive file=iscsi://127.0.0.1/iqn.qemu.test/1 \
    -readconfig iscsi.conf
```

Howto set up a simple iSCSI target on loopback and accessing it via QEMU:

```
This example shows how to set up an iSCSI target with one CDROM and one DISK
using the Linux STGT software target. This target is available on Red Hat based
systems as the package 'scsi-target-utils'.

tgtd --iscsi portal=127.0.0.1:3260
tgtadm --lld iscsi --op new --mode target --tid 1 -T iqn.qemu.test
tgtadm --lld iscsi --mode logicalunit --op new --tid 1 --lun 1 \
    -b /IMAGES/disk.img --device-type=disk
tgtadm --lld iscsi --mode logicalunit --op new --tid 1 --lun 2 \
    -b /IMAGES/cd.iso --device-type=cd
tgtadm --lld iscsi --op bind --mode target --tid 1 -I ALL

qemu-system-i386 -iscsi initiator-name=iqn.qemu.test:my-initiator \
    -boot d -drive file=iscsi://127.0.0.1/iqn.qemu.test/1 \
    -cdrom iscsi://127.0.0.1/iqn.qemu.test/2
```

2.7.13 GlusterFS disk images

GlusterFS is an user space distributed file system.

You can boot from the GlusterFS disk image with the command:

```
URI:
qemu-system-x86_64 -drive file=gluster[+type]://[host[:port]]/volume/path
                                [?socket=...][,file.debug=9][,file.logfile=...]

JSON:
qemu-system-x86_64 'json:{"driver":"qcow2",
                           "file":{"driver":"gluster",
                                   "volume":"testvol","path":"a.img","debug":9,"logfile":"
                                   "server":[{"type":"tcp","host":"...","port":"..."},
                                             {"type":"unix","socket":"..."}]}}'
```

gluster is the protocol.

type specifies the transport type used to connect to gluster management daemon (glusterd). Valid transport types are tcp and unix. In the URI form, if a transport type isn't specified, then tcp type is assumed.

host specifies the server where the volume file specification for the given volume resides. This can be either a hostname or an ipv4 address. If transport type is unix, then *host* field

should not be specified. Instead *socket* field needs to be populated with the path to unix domain socket.

port is the port number on which glusterd is listening. This is optional and if not specified, it defaults to port 24007. If the transport type is unix, then *port* should not be specified.

volume is the name of the gluster volume which contains the disk image.

path is the path to the actual disk image that resides on gluster volume.

debug is the logging level of the gluster protocol driver. Debug levels are 0-9, with 9 being the most verbose, and 0 representing no debugging output. The default level is 4. The current logging levels defined in the gluster source are 0 - None, 1 - Emergency, 2 - Alert, 3 - Critical, 4 - Error, 5 - Warning, 6 - Notice, 7 - Info, 8 - Debug, 9 - Trace

logfile is a commandline option to mention log file path which helps in logging to the specified file and also help in persisting the gfapi logs. The default is stderr.

You can create a GlusterFS disk image with the command:

`qemu-img create gluster://host/volume/path size`

Examples

```
qemu-system-x86_64 -drive file=gluster://1.2.3.4/testvol/a.img
qemu-system-x86_64 -drive file=gluster+tcp://1.2.3.4/testvol/a.img
qemu-system-x86_64 -drive file=gluster+tcp://1.2.3.4:24007/testvol/dir/a.img
qemu-system-x86_64 -drive file=gluster+tcp://[1:2:3:4:5:6:7:8]/testvol/dir/a.img
qemu-system-x86_64 -drive file=gluster+tcp://[1:2:3:4:5:6:7:8]:24007/testvol/dir/a.img
qemu-system-x86_64 -drive file=gluster+tcp://server.domain.com:24007/testvol/dir/a.img
qemu-system-x86_64 -drive file=gluster+unix:///testvol/dir/a.img?socket=/tmp/glusterd.socke
qemu-system-x86_64 -drive file=gluster+rdma://1.2.3.4:24007/testvol/a.img
qemu-system-x86_64 -drive file=gluster://1.2.3.4/testvol/a.img,file.debug=9,file.logfile=/v
qemu-system-x86_64 'json:{"driver":"qcow2",
                          "file":{"driver":"gluster",
                                  "volume":"testvol","path":"a.img",
                                  "debug":9,"logfile":"/var/log/qemu-gluster.log",
                                  "server":[{"type":"tcp","host":"1.2.3.4","port":24007},
                                            {"type":"unix","socket":"/var/run/glusterd.so
qemu-system-x86_64 -drive driver=qcow2,file.driver=gluster,file.volume=testvol,file.path=/p
                                     file.debug=9,file.logfile=/var/log/qemu-gluster.log,
                                     file.server.0.type=tcp,file.server.0.host=1.2.3.4,fi
                                     file.server.1.type=unix,file.server.1.socket=/var/ru
```

2.7.14 Secure Shell (ssh) disk images

You can access disk images located on a remote ssh server by using the ssh protocol:

`qemu-system-x86_64 -drive file=ssh://[user@]server[:port]/path[?host_key_check=host_key_che`

Alternative syntax using properties:

`qemu-system-x86_64 -drive file.driver=ssh[,file.user=user],file.host=server[,file.port=port`

ssh is the protocol.

user is the remote user. If not specified, then the local username is tried.

server specifies the remote ssh server. Any ssh server can be used, but it must implement the sftp-server protocol. Most Unix/Linux systems should work without requiring any extra configuration.

port is the port number on which sshd is listening. By default the standard ssh port (22) is used.

path is the path to the disk image.

The optional *host_key_check* parameter controls how the remote host's key is checked. The default is `yes` which means to use the local `.ssh/known_hosts` file. Setting this to `no` turns off known-hosts checking. Or you can check that the host key matches a specific fingerprint: `host_key_check=md5:78:45:8e:14:57:4f:d5:45:83:0a:0e:f3:49:82:c9:c8` (`sha1:` can also be used as a prefix, but note that OpenSSH tools only use MD5 to print fingerprints).

Currently authentication must be done using ssh-agent. Other authentication methods may be supported in future.

Note: Many ssh servers do not support an `fsync`-style operation. The ssh driver cannot guarantee that disk flush requests are obeyed, and this causes a risk of disk corruption if the remote server or network goes down during writes. The driver will print a warning when `fsync` is not supported:

warning: ssh server `ssh.example.com:22` does not support fsync

With sufficiently new versions of libssh2 and OpenSSH, `fsync` is supported.

2.8 Network emulation

QEMU can simulate several network cards (PCI or ISA cards on the PC target) and can connect them to an arbitrary number of Virtual Local Area Networks (VLANs). Host TAP devices can be connected to any QEMU VLAN. VLAN can be connected between separate instances of QEMU to simulate large networks. For simpler usage, a non privileged user mode network stack can replace the TAP device to have a basic network connection.

2.8.1 VLANs

QEMU simulates several VLANs. A VLAN can be symbolised as a virtual connection between several network devices. These devices can be for example QEMU virtual Ethernet cards or virtual Host ethernet devices (TAP devices).

2.8.2 Using TAP network interfaces

This is the standard way to connect QEMU to a real network. QEMU adds a virtual network device on your host (called `tapN`), and you can then configure it as if it was a real ethernet card.

2.8.2.1 Linux host

As an example, you can download the `linux-test-xxx.tar.gz` archive and copy the script `qemu-ifup` in `/etc` and configure properly `sudo` so that the command `ifconfig` contained in `qemu-ifup` can be executed as root. You must verify that your host kernel supports the TAP network interfaces: the device `/dev/net/tun` must be present.

See Section 2.3 [sec_invocation], page 3, to have examples of command lines using the TAP network interfaces.

2.8.2.2 Windows host

There is a virtual ethernet driver for Windows 2000/XP systems, called TAP-Win32. But it is not included in standard QEMU for Windows, so you will need to get it separately. It is part of OpenVPN package, so download OpenVPN from : `http://openvpn.net/`.

2.8.3 Using the user mode network stack

By using the option `-net user` (default configuration if no `-net` option is specified), QEMU uses a completely user mode network stack (you don't need root privilege to use the virtual network). The virtual network configuration is the following:

```
QEMU VLAN      <------>  Firewall/DHCP server <-----> Internet
                   |          (10.0.2.2)
                   |
                   ---->  DNS server (10.0.2.3)
                   |
                   ---->  SMB server (10.0.2.4)
```

The QEMU VM behaves as if it was behind a firewall which blocks all incoming connections. You can use a DHCP client to automatically configure the network in the QEMU VM. The DHCP server assign addresses to the hosts starting from 10.0.2.15.

In order to check that the user mode network is working, you can ping the address 10.0.2.2 and verify that you got an address in the range 10.0.2.x from the QEMU virtual DHCP server.

Note that ICMP traffic in general does not work with user mode networking. `ping`, aka. ICMP echo, to the local router (10.0.2.2) shall work, however. If you're using QEMU on Linux >= 3.0, it can use unprivileged ICMP ping sockets to allow `ping` to the Internet. The host admin has to set the ping_group_range in order to grant access to those sockets. To allow ping for GID 100 (usually users group):

```
echo 100 100 > /proc/sys/net/ipv4/ping_group_range
```

When using the built-in TFTP server, the router is also the TFTP server.

When using the `'-netdev user,hostfwd=...'` option, TCP or UDP connections can be redirected from the host to the guest. It allows for example to redirect X11, telnet or SSH connections.

2.8.4 Connecting VLANs between QEMU instances

Using the `-net socket` option, it is possible to make VLANs that span several QEMU instances. See Section 2.3 [sec_invocation], page 3, to have a basic example.

2.9 Other Devices

2.9.1 Inter-VM Shared Memory device

On Linux hosts, a shared memory device is available. The basic syntax is:

```
qemu-system-x86_64 -device ivshmem-plain,memdev=hostmem
```

where *hostmem* names a host memory backend. For a POSIX shared memory backend, use something like

```
-object memory-backend-file,size=1M,share,mem-path=/dev/shm/ivshmem,id=hostmem
```

If desired, interrupts can be sent between guest VMs accessing the same shared memory region. Interrupt support requires using a shared memory server and using a chardev socket to connect to it. The code for the shared memory server is qemu.git/contrib/ivshmem-server. An example syntax when using the shared memory server is:

```
# First start the ivshmem server once and for all
ivshmem-server -p pidfile -S path -m shm-name -l shm-size -n vectors

# Then start your qemu instances with matching arguments
qemu-system-x86_64 -device ivshmem-doorbell,vectors=vectors,chardev=id
                   -chardev socket,path=path,id=id
```

When using the server, the guest will be assigned a VM ID (>=0) that allows guests using the same server to communicate via interrupts. Guests can read their VM ID from a device register (see ivshmem-spec.txt).

2.9.1.1 Migration with ivshmem

With device property `master=on`, the guest will copy the shared memory on migration to the destination host. With `master=off`, the guest will not be able to migrate with the device attached. In the latter case, the device should be detached and then reattached after migration using the PCI hotplug support.

At most one of the devices sharing the same memory can be master. The master must complete migration before you plug back the other devices.

2.9.1.2 ivshmem and hugepages

Instead of specifying the <shm size> using POSIX shm, you may specify a memory backend that has hugepage support:

```
qemu-system-x86_64 -object memory-backend-file,size=1G,mem-path=/dev/hugepages/my-shmem-fil
                   -device ivshmem-plain,memdev=mb1
```

ivshmem-server also supports hugepages mount points with the `-m` memory path argument.

2.10 Direct Linux Boot

This section explains how to launch a Linux kernel inside QEMU without having to make a full bootable image. It is very useful for fast Linux kernel testing.

The syntax is:

```
qemu-system-i386 -kernel arch/i386/boot/bzImage -hda root-2.4.20.img -append "root=/dev/hda
```

Use `-kernel` to provide the Linux kernel image and `-append` to give the kernel command line arguments. The `-initrd` option can be used to provide an INITRD image.

When using the direct Linux boot, a disk image for the first hard disk `hda` is required because its boot sector is used to launch the Linux kernel.

If you do not need graphical output, you can disable it and redirect the virtual serial port and the QEMU monitor to the console with the `-nographic` option. The typical command line is:

```
qemu-system-i386 -kernel arch/i386/boot/bzImage -hda root-2.4.20.img \
```

```
-append "root=/dev/hda console=ttyS0" -nographic
```

Use `Ctrl-a c` to switch between the serial console and the monitor (see Section 2.4 [pc-sys_keys], page 52).

2.11 USB emulation

QEMU emulates a PCI UHCI USB controller. You can virtually plug virtual USB devices or real host USB devices (experimental, works only on Linux hosts). QEMU will automatically create and connect virtual USB hubs as necessary to connect multiple USB devices.

2.11.1 Connecting USB devices

USB devices can be connected with the `-usbdevice` commandline option or the `usb_add` monitor command. Available devices are:

`mouse` Virtual Mouse. This will override the PS/2 mouse emulation when activated.

`tablet` Pointer device that uses absolute coordinates (like a touchscreen). This means QEMU is able to report the mouse position without having to grab the mouse. Also overrides the PS/2 mouse emulation when activated.

`disk:`*file*
: Mass storage device based on *file* (see Section 2.7 [disk_images], page 64)

`host:`*bus.addr*
: Pass through the host device identified by *bus.addr* (Linux only)

`host:`*vendor_id:product_id*
: Pass through the host device identified by *vendor_id:product_id* (Linux only)

`wacom-tablet`
: Virtual Wacom PenPartner tablet. This device is similar to the `tablet` above but it can be used with the tslib library because in addition to touch coordinates it reports touch pressure.

`keyboard` Standard USB keyboard. Will override the PS/2 keyboard (if present).

`serial:[vendorid=`*vendor_id*`][,product_id=`*product_id*`]:`*dev*
: Serial converter. This emulates an FTDI FT232BM chip connected to host character device *dev*. The available character devices are the same as for the `-serial` option. The `vendorid` and `productid` options can be used to override the default 0403:6001. For instance,

  ```
  usb_add serial:productid=FA00:tcp:192.168.0.2:4444
  ```

  will connect to tcp port 4444 of ip 192.168.0.2, and plug that to the virtual serial converter, faking a Matrix Orbital LCD Display (USB ID 0403:FA00).

`braille` Braille device. This will use BrlAPI to display the braille output on a real or fake device.

`net:`*options*
: Network adapter that supports CDC ethernet and RNDIS protocols. *options* specifies NIC options as with `-net nic,`*options* (see description). For instance, user-mode networking can be used with

  ```
  qemu-system-i386 [...OPTIONS...] -net user,vlan=0 -usbdevice net:vlan=0
  ```

Currently this cannot be used in machines that support PCI NICs.

`bt[:hci-type]`
: Bluetooth dongle whose type is specified in the same format as with the `-bt hci` option, see [allowed HCI types], page 36. If no type is given, the HCI logic corresponds to `-bt hci,vlan=0`. This USB device implements the USB Transport Layer of HCI. Example usage:

```
qemu-system-i386 [...OPTIONS...] -usbdevice bt:hci,vlan=3 -bt device:keyboard,vla
```

2.11.2 Using host USB devices on a Linux host

WARNING: this is an experimental feature. QEMU will slow down when using it. USB devices requiring real time streaming (i.e. USB Video Cameras) are not supported yet.

1. If you use an early Linux 2.4 kernel, verify that no Linux driver is actually using the USB device. A simple way to do that is simply to disable the corresponding kernel module by renaming it from `mydriver.o` to `mydriver.o.disabled`.
2. Verify that `/proc/bus/usb` is working (most Linux distributions should enable it by default). You should see something like that:

   ```
   ls /proc/bus/usb
   001  devices  drivers
   ```
3. Since only root can access to the USB devices directly, you can either launch QEMU as root or change the permissions of the USB devices you want to use. For testing, the following suffices:

   ```
   chown -R myuid /proc/bus/usb
   ```
4. Launch QEMU and do in the monitor:

   ```
   info usbhost
     Device 1.2, speed 480 Mb/s
       Class 00: USB device 1234:5678, USB DISK
   ```

   You should see the list of the devices you can use (Never try to use hubs, it won't work).
5. Add the device in QEMU by using:

   ```
   usb_add host:1234:5678
   ```

   Normally the guest OS should report that a new USB device is plugged. You can use the option `-usbdevice` to do the same.
6. Now you can try to use the host USB device in QEMU.

When relaunching QEMU, you may have to unplug and plug again the USB device to make it work again (this is a bug).

2.12 VNC security

The VNC server capability provides access to the graphical console of the guest VM across the network. This has a number of security considerations depending on the deployment scenarios.

2.12.1 Without passwords

The simplest VNC server setup does not include any form of authentication. For this setup it is recommended to restrict it to listen on a UNIX domain socket only. For example

```
qemu-system-i386 [...OPTIONS...] -vnc unix:/home/joebloggs/.qemu-myvm-vnc
```

This ensures that only users on local box with read/write access to that path can access the VNC server. To securely access the VNC server from a remote machine, a combination of netcat+ssh can be used to provide a secure tunnel.

2.12.2 With passwords

The VNC protocol has limited support for password based authentication. Since the protocol limits passwords to 8 characters it should not be considered to provide high security. The password can be fairly easily brute-forced by a client making repeat connections. For this reason, a VNC server using password authentication should be restricted to only listen on the loopback interface or UNIX domain sockets. Password authentication is not supported when operating in FIPS 140-2 compliance mode as it requires the use of the DES cipher. Password authentication is requested with the `password` option, and then once QEMU is running the password is set with the monitor. Until the monitor is used to set the password all clients will be rejected.

```
qemu-system-i386 [...OPTIONS...] -vnc :1,password -monitor stdio
(qemu) change vnc password
Password: ********
(qemu)
```

2.12.3 With x509 certificates

The QEMU VNC server also implements the VeNCrypt extension allowing use of TLS for encryption of the session, and x509 certificates for authentication. The use of x509 certificates is strongly recommended, because TLS on its own is susceptible to man-in-the-middle attacks. Basic x509 certificate support provides a secure session, but no authentication. This allows any client to connect, and provides an encrypted session.

```
qemu-system-i386 [...OPTIONS...] -vnc :1,tls,x509=/etc/pki/qemu -monitor stdio
```

In the above example `/etc/pki/qemu` should contain at least three files, `ca-cert.pem`, `server-cert.pem` and `server-key.pem`. Unprivileged users will want to use a private directory, for example `$HOME/.pki/qemu`. NB the `server-key.pem` file should be protected with file mode 0600 to only be readable by the user owning it.

2.12.4 With x509 certificates and client verification

Certificates can also provide a means to authenticate the client connecting. The server will request that the client provide a certificate, which it will then validate against the CA certificate. This is a good choice if deploying in an environment with a private internal certificate authority.

```
qemu-system-i386 [...OPTIONS...] -vnc :1,tls,x509verify=/etc/pki/qemu -monitor stdio
```

2.12.5 With x509 certificates, client verification and passwords

Finally, the previous method can be combined with VNC password authentication to provide two layers of authentication for clients.

```
qemu-system-i386 [...OPTIONS...] -vnc :1,password,tls,x509verify=/etc/pki/qemu -monitor std
(qemu) change vnc password
Password: ********
(qemu)
```

2.12.6 With SASL authentication

The SASL authentication method is a VNC extension, that provides an easily extendable, pluggable authentication method. This allows for integration with a wide range of authentication mechanisms, such as PAM, GSSAPI/Kerberos, LDAP, SQL databases, one-time keys and more. The strength of the authentication depends on the exact mechanism configured. If the chosen mechanism also provides a SSF layer, then it will encrypt the datastream as well.

Refer to the later docs on how to choose the exact SASL mechanism used for authentication, but assuming use of one supporting SSF, then QEMU can be launched with:

```
qemu-system-i386 [...OPTIONS...] -vnc :1,sasl -monitor stdio
```

2.12.7 With x509 certificates and SASL authentication

If the desired SASL authentication mechanism does not supported SSF layers, then it is strongly advised to run it in combination with TLS and x509 certificates. This provides securely encrypted data stream, avoiding risk of compromising of the security credentials. This can be enabled, by combining the 'sasl' option with the aforementioned TLS + x509 options:

```
qemu-system-i386 [...OPTIONS...] -vnc :1,tls,x509,sasl -monitor stdio
```

2.12.8 Generating certificates for VNC

The GNU TLS packages provides a command called `certtool` which can be used to generate certificates and keys in PEM format. At a minimum it is necessary to setup a certificate authority, and issue certificates to each server. If using certificates for authentication, then each client will also need to be issued a certificate. The recommendation is for the server to keep its certificates in either `/etc/pki/qemu` or for unprivileged users in `$HOME/.pki/qemu`.

2.12.8.1 Setup the Certificate Authority

This step only needs to be performed once per organization / organizational unit. First the CA needs a private key. This key must be kept VERY secret and secure. If this key is compromised the entire trust chain of the certificates issued with it is lost.

```
# certtool --generate-privkey > ca-key.pem
```

A CA needs to have a public certificate. For simplicity it can be a self-signed certificate, or one issue by a commercial certificate issuing authority. To generate a self-signed certificate requires one core piece of information, the name of the organization.

```
# cat > ca.info <<EOF
cn = Name of your organization
ca
cert_signing_key
EOF
# certtool --generate-self-signed \
```

```
            --load-privkey ca-key.pem
            --template ca.info \
            --outfile ca-cert.pem
```

The `ca-cert.pem` file should be copied to all servers and clients wishing to utilize TLS support in the VNC server. The `ca-key.pem` must not be disclosed/copied at all.

2.12.8.2 Issuing server certificates

Each server (or host) needs to be issued with a key and certificate. When connecting the certificate is sent to the client which validates it against the CA certificate. The core piece of information for a server certificate is the hostname. This should be the fully qualified hostname that the client will connect with, since the client will typically also verify the hostname in the certificate. On the host holding the secure CA private key:

```
# cat > server.info <<EOF
organization = Name  of your organization
cn = server.foo.example.com
tls_www_server
encryption_key
signing_key
EOF
# certtool --generate-privkey > server-key.pem
# certtool --generate-certificate \
            --load-ca-certificate ca-cert.pem \
            --load-ca-privkey ca-key.pem \
            --load-privkey server-key.pem \
            --template server.info \
            --outfile server-cert.pem
```

The `server-key.pem` and `server-cert.pem` files should now be securely copied to the server for which they were generated. The `server-key.pem` is security sensitive and should be kept protected with file mode 0600 to prevent disclosure.

2.12.8.3 Issuing client certificates

If the QEMU VNC server is to use the `x509verify` option to validate client certificates as its authentication mechanism, each client also needs to be issued a certificate. The client certificate contains enough metadata to uniquely identify the client, typically organization, state, city, building, etc. On the host holding the secure CA private key:

```
# cat > client.info <<EOF
country = GB
state = London
locality = London
organization = Name of your organization
cn = client.foo.example.com
tls_www_client
encryption_key
signing_key
EOF
# certtool --generate-privkey > client-key.pem
```

```
# certtool --generate-certificate \
           --load-ca-certificate ca-cert.pem \
           --load-ca-privkey ca-key.pem \
           --load-privkey client-key.pem \
           --template client.info \
           --outfile client-cert.pem
```

The `client-key.pem` and `client-cert.pem` files should now be securely copied to the client for which they were generated.

2.12.9 Configuring SASL mechanisms

The following documentation assumes use of the Cyrus SASL implementation on a Linux host, but the principals should apply to any other SASL impl. When SASL is enabled, the mechanism configuration will be loaded from system default SASL service config /etc/sasl2/qemu.conf. If running QEMU as an unprivileged user, an environment variable SASL_CONF_PATH can be used to make it search alternate locations for the service config.

The default configuration might contain

```
mech_list: digest-md5
sasldb_path: /etc/qemu/passwd.db
```

This says to use the 'Digest MD5' mechanism, which is similar to the HTTP Digest-MD5 mechanism. The list of valid usernames & passwords is maintained in the /etc/qemu/passwd.db file, and can be updated using the saslpasswd2 command. While this mechanism is easy to configure and use, it is not considered secure by modern standards, so only suitable for developers / ad-hoc testing.

A more serious deployment might use Kerberos, which is done with the 'gssapi' mechanism

```
mech_list: gssapi
keytab: /etc/qemu/krb5.tab
```

For this to work the administrator of your KDC must generate a Kerberos principal for the server, with a name of 'qemu/somehost.example.com@EXAMPLE.COM' replacing 'somehost.example.com' with the fully qualified host name of the machine running QEMU, and 'EXAMPLE.COM' with the Kerberos Realm.

Other configurations will be left as an exercise for the reader. It should be noted that only Digest-MD5 and GSSAPI provides a SSF layer for data encryption. For all other mechanisms, VNC should always be configured to use TLS and x509 certificates to protect security credentials from snooping.

2.13 GDB usage

QEMU has a primitive support to work with gdb, so that you can do 'Ctrl-C' while the virtual machine is running and inspect its state.

In order to use gdb, launch QEMU with the '-s' option. It will wait for a gdb connection:

```
qemu-system-i386 -s -kernel arch/i386/boot/bzImage -hda root-2.4.20.img \
                    -append "root=/dev/hda"
Connected to host network interface: tun0
Waiting gdb connection on port 1234
```

Then launch gdb on the 'vmlinux' executable:

```
> gdb vmlinux
```

In gdb, connect to QEMU:

```
(gdb) target remote localhost:1234
```

Then you can use gdb normally. For example, type 'c' to launch the kernel:

```
(gdb) c
```

Here are some useful tips in order to use gdb on system code:

1. Use `info reg` to display all the CPU registers.
2. Use `x/10i $eip` to display the code at the PC position.
3. Use `set architecture i8086` to dump 16 bit code. Then use `x/10i $cs*16+$eip` to dump the code at the PC position.

Advanced debugging options:

The default single stepping behavior is step with the IRQs and timer service routines off. It is set this way because when gdb executes a single step it expects to advance beyond the current instruction. With the IRQs and timer service routines on, a single step might jump into the one of the interrupt or exception vectors instead of executing the current instruction. This means you may hit the same breakpoint a number of times before executing the instruction gdb wants to have executed. Because there are rare circumstances where you want to single step into an interrupt vector the behavior can be controlled from GDB. There are three commands you can query and set the single step behavior:

`maintenance packet qqemu.sstepbits`
: This will display the MASK bits used to control the single stepping IE:

```
(gdb) maintenance packet qqemu.sstepbits
sending: "qqemu.sstepbits"
received: "ENABLE=1,NOIRQ=2,NOTIMER=4"
```

`maintenance packet qqemu.sstep`
: This will display the current value of the mask used when single stepping IE:

```
(gdb) maintenance packet qqemu.sstep
sending: "qqemu.sstep"
received: "0x7"
```

`maintenance packet Qqemu.sstep=HEX_VALUE`
: This will change the single step mask, so if wanted to enable IRQs on the single step, but not timers, you would use:

```
(gdb) maintenance packet Qqemu.sstep=0x5
sending: "qemu.sstep=0x5"
received: "OK"
```

2.14 Target OS specific information

2.14.1 Linux

To have access to SVGA graphic modes under X11, use the `vesa` or the `cirrus` X11 driver. For optimal performances, use 16 bit color depth in the guest and the host OS.

When using a 2.6 guest Linux kernel, you should add the option `clock=pit` on the kernel command line because the 2.6 Linux kernels make very strict real time clock checks by default that QEMU cannot simulate exactly.

When using a 2.6 guest Linux kernel, verify that the 4G/4G patch is not activated because QEMU is slower with this patch. The QEMU Accelerator Module is also much slower in this case. Earlier Fedora Core 3 Linux kernel (< 2.6.9-1.724_FC3) were known to incorporate this patch by default. Newer kernels don't have it.

2.14.2 Windows

If you have a slow host, using Windows 95 is better as it gives the best speed. Windows 2000 is also a good choice.

2.14.2.1 SVGA graphic modes support

QEMU emulates a Cirrus Logic GD5446 Video card. All Windows versions starting from Windows 95 should recognize and use this graphic card. For optimal performances, use 16 bit color depth in the guest and the host OS.

If you are using Windows XP as guest OS and if you want to use high resolution modes which the Cirrus Logic BIOS does not support (i.e. >= 1280x1024x16), then you should use the VESA VBE virtual graphic card (option `-std-vga`).

2.14.2.2 CPU usage reduction

Windows 9x does not correctly use the CPU HLT instruction. The result is that it takes host CPU cycles even when idle. You can install the utility from `http://www.user.cityline.ru/~maxamn/amnhltm.zip` to solve this problem. Note that no such tool is needed for NT, 2000 or XP.

2.14.2.3 Windows 2000 disk full problem

Windows 2000 has a bug which gives a disk full problem during its installation. When installing it, use the `-win2k-hack` QEMU option to enable a specific workaround. After Windows 2000 is installed, you no longer need this option (this option slows down the IDE transfers).

2.14.2.4 Windows 2000 shutdown

Windows 2000 cannot automatically shutdown in QEMU although Windows 98 can. It comes from the fact that Windows 2000 does not automatically use the APM driver provided by the BIOS.

In order to correct that, do the following (thanks to Struan Bartlett): go to the Control Panel => Add/Remove Hardware & Next => Add/Troubleshoot a device => Add a new device & Next => No, select the hardware from a list & Next => NT Apm/Legacy Support & Next => Next (again) a few times. Now the driver is installed and Windows 2000 now correctly instructs QEMU to shutdown at the appropriate moment.

2.14.2.5 Share a directory between Unix and Windows

See Section 2.3 [sec_invocation], page 3, about the help of the option '`-netdev user,smb=...`'.

2.14.2.6 Windows XP security problem

Some releases of Windows XP install correctly but give a security error when booting:

```
A problem is preventing Windows from accurately checking the
license for this computer. Error code: 0x800703e6.
```

The workaround is to install a service pack for XP after a boot in safe mode. Then reboot, and the problem should go away. Since there is no network while in safe mode, its recommended to download the full installation of SP1 or SP2 and transfer that via an ISO or using the vvfat block device ("-hdb fat:directory_which_holds_the_SP").

2.14.3 MS-DOS and FreeDOS

2.14.3.1 CPU usage reduction

DOS does not correctly use the CPU HLT instruction. The result is that it takes host CPU cycles even when idle. You can install the utility from `http://www.vmware.com/software/dosidle210.zip` to solve this problem.

3 QEMU System emulator for non PC targets

QEMU is a generic emulator and it emulates many non PC machines. Most of the options are similar to the PC emulator. The differences are mentioned in the following sections.

3.1 PowerPC System emulator

Use the executable `qemu-system-ppc` to simulate a complete PREP or PowerMac PowerPC system.

QEMU emulates the following PowerMac peripherals:

- UniNorth or Grackle PCI Bridge
- PCI VGA compatible card with VESA Bochs Extensions
- 2 PMAC IDE interfaces with hard disk and CD-ROM support
- NE2000 PCI adapters
- Non Volatile RAM
- VIA-CUDA with ADB keyboard and mouse.

QEMU emulates the following PREP peripherals:

- PCI Bridge
- PCI VGA compatible card with VESA Bochs Extensions
- 2 IDE interfaces with hard disk and CD-ROM support
- Floppy disk
- NE2000 network adapters
- Serial port
- PREP Non Volatile RAM
- PC compatible keyboard and mouse.

QEMU uses the Open Hack'Ware Open Firmware Compatible BIOS available at `http://perso.magic.fr/l_indien/OpenHackWare/index.htm`.

Since version 0.9.1, QEMU uses OpenBIOS `http://www.openbios.org/` for the g3beige and mac99 PowerMac machines. OpenBIOS is a free (GPL v2) portable firmware implementation. The goal is to implement a 100% IEEE 1275-1994 (referred to as Open Firmware) compliant firmware.

The following options are specific to the PowerPC emulation:

`-g` *`WxH`*`[x`*`DEPTH`*`]`
: Set the initial VGA graphic mode. The default is 800x600x32.

`-prom-env` *`string`*
: Set OpenBIOS variables in NVRAM, for example:

```
qemu-system-ppc -prom-env 'auto-boot?=false' \
 -prom-env 'boot-device=hd:2,\yaboot' \
 -prom-env 'boot-args=conf=hd:2,\yaboot.conf'
```

: These variables are not used by Open Hack'Ware.

More information is available at `http://perso.magic.fr/l_indien/qemu-ppc/`.

3.2 Sparc32 System emulator

Use the executable `qemu-system-sparc` to simulate the following Sun4m architecture machines:

- SPARCstation 4
- SPARCstation 5
- SPARCstation 10
- SPARCstation 20
- SPARCserver 600MP
- SPARCstation LX
- SPARCstation Voyager
- SPARCclassic
- SPARCbook

The emulation is somewhat complete. SMP up to 16 CPUs is supported, but Linux limits the number of usable CPUs to 4.

QEMU emulates the following sun4m peripherals:

- IOMMU
- TCX or cgthree Frame buffer
- Lance (Am7990) Ethernet
- Non Volatile RAM M48T02/M48T08
- Slave I/O: timers, interrupt controllers, Zilog serial ports, keyboard and power/reset logic
- ESP SCSI controller with hard disk and CD-ROM support
- Floppy drive (not on SS-600MP)
- CS4231 sound device (only on SS-5, not working yet)

The number of peripherals is fixed in the architecture. Maximum memory size depends on the machine type, for SS-5 it is 256MB and for others 2047MB.

Since version 0.8.2, QEMU uses OpenBIOS `http://www.openbios.org/`. OpenBIOS is a free (GPL v2) portable firmware implementation. The goal is to implement a 100% IEEE 1275-1994 (referred to as Open Firmware) compliant firmware.

A sample Linux 2.6 series kernel and ram disk image are available on the QEMU web site. There are still issues with NetBSD and OpenBSD, but most kernel versions work. Please note that currently older Solaris kernels don't work probably due to interface issues between OpenBIOS and Solaris.

The following options are specific to the Sparc32 emulation:

`-g` *`WxHx`*`[x`*`DEPTH`*`]`
: Set the initial graphics mode. For TCX, the default is 1024x768x8 with the option of 1024x768x24. For cgthree, the default is 1024x768x8 with the option of 1152x900x8 for people who wish to use OBP.

`-prom-env` *`string`*
: Set OpenBIOS variables in NVRAM, for example:

```
qemu-system-sparc -prom-env 'auto-boot?=false' \
```

```
-prom-env 'boot-device=sd(0,2,0):d' -prom-env 'boot-args=linux single'
```

`-M [SS-4|SS-5|SS-10|SS-20|SS-600MP|LX|Voyager|SPARCClassic] [|SPARCbook]`
Set the emulated machine type. Default is SS-5.

3.3 Sparc64 System emulator

Use the executable `qemu-system-sparc64` to simulate a Sun4u (UltraSPARC PC-like machine), Sun4v (T1 PC-like machine), or generic Niagara (T1) machine. The Sun4u emulator is mostly complete, being able to run Linux, NetBSD and OpenBSD in headless (-nographic) mode. The Sun4v and Niagara emulators are still a work in progress.

QEMU emulates the following peripherals:

- UltraSparc IIi APB PCI Bridge
- PCI VGA compatible card with VESA Bochs Extensions
- PS/2 mouse and keyboard
- Non Volatile RAM M48T59
- PC-compatible serial ports
- 2 PCI IDE interfaces with hard disk and CD-ROM support
- Floppy disk

The following options are specific to the Sparc64 emulation:

`-prom-env` *`string`*
Set OpenBIOS variables in NVRAM, for example:

```
qemu-system-sparc64 -prom-env 'auto-boot?=false'
```

`-M [sun4u|sun4v|Niagara]`
Set the emulated machine type. The default is sun4u.

3.4 MIPS System emulator

Four executables cover simulation of 32 and 64-bit MIPS systems in both endian options, `qemu-system-mips`, `qemu-system-mipsel` `qemu-system-mips64` and `qemu-system-mips64el`. Five different machine types are emulated:

- A generic ISA PC-like machine `"mips"`
- The MIPS Malta prototype board `"malta"`
- An ACER Pica `"pica61"`. This machine needs the 64-bit emulator.
- MIPS emulator pseudo board `"mipssim"`
- A MIPS Magnum R4000 machine `"magnum"`. This machine needs the 64-bit emulator.

The generic emulation is supported by Debian 'Etch' and is able to install Debian into a virtual disk image. The following devices are emulated:

- A range of MIPS CPUs, default is the 24Kf
- PC style serial port
- PC style IDE disk
- NE2000 network card

The Malta emulation supports the following devices:

- Core board with MIPS 24Kf CPU and Galileo system controller
- PIIX4 PCI/USB/SMbus controller
- The Multi-I/O chip's serial device
- PCI network cards (PCnet32 and others)
- Malta FPGA serial device
- Cirrus (default) or any other PCI VGA graphics card

The ACER Pica emulation supports:

- MIPS R4000 CPU
- PC-style IRQ and DMA controllers
- PC Keyboard
- IDE controller

The mipssim pseudo board emulation provides an environment similar to what the proprietary MIPS emulator uses for running Linux. It supports:

- A range of MIPS CPUs, default is the 24Kf
- PC style serial port
- MIPSnet network emulation

The MIPS Magnum R4000 emulation supports:

- MIPS R4000 CPU
- PC-style IRQ controller
- PC Keyboard
- SCSI controller
- G364 framebuffer

3.5 ARM System emulator

Use the executable `qemu-system-arm` to simulate a ARM machine. The ARM Integrator/CP board is emulated with the following devices:

- ARM926E, ARM1026E, ARM946E, ARM1136 or Cortex-A8 CPU
- Two PL011 UARTs
- SMC 91c111 Ethernet adapter
- PL110 LCD controller
- PL050 KMI with PS/2 keyboard and mouse.
- PL181 MultiMedia Card Interface with SD card.

The ARM Versatile baseboard is emulated with the following devices:

- ARM926E, ARM1136 or Cortex-A8 CPU
- PL190 Vectored Interrupt Controller
- Four PL011 UARTs
- SMC 91c111 Ethernet adapter

- PL110 LCD controller
- PL050 KMI with PS/2 keyboard and mouse.
- PCI host bridge. Note the emulated PCI bridge only provides access to PCI memory space. It does not provide access to PCI IO space. This means some devices (eg. ne2k_pci NIC) are not usable, and others (eg. rtl8139 NIC) are only usable when the guest drivers use the memory mapped control registers.
- PCI OHCI USB controller.
- LSI53C895A PCI SCSI Host Bus Adapter with hard disk and CD-ROM devices.
- PL181 MultiMedia Card Interface with SD card.

Several variants of the ARM RealView baseboard are emulated, including the EB, PB-A8 and PBX-A9. Due to interactions with the bootloader, only certain Linux kernel configurations work out of the box on these boards.

Kernels for the PB-A8 board should have CONFIG_REALVIEW_HIGH_PHYS_OFFSET enabled in the kernel, and expect 512M RAM. Kernels for The PBX-A9 board should have CONFIG_SPARSEMEM enabled, CONFIG_REALVIEW_HIGH_PHYS_OFFSET disabled and expect 1024M RAM.

The following devices are emulated:

- ARM926E, ARM1136, ARM11MPCore, Cortex-A8 or Cortex-A9 MPCore CPU
- ARM AMBA Generic/Distributed Interrupt Controller
- Four PL011 UARTs
- SMC 91c111 or SMSC LAN9118 Ethernet adapter
- PL110 LCD controller
- PL050 KMI with PS/2 keyboard and mouse
- PCI host bridge
- PCI OHCI USB controller
- LSI53C895A PCI SCSI Host Bus Adapter with hard disk and CD-ROM devices
- PL181 MultiMedia Card Interface with SD card.

The XScale-based clamshell PDA models ("Spitz", "Akita", "Borzoi" and "Terrier") emulation includes the following peripherals:

- Intel PXA270 System-on-chip (ARM V5TE core)
- NAND Flash memory
- IBM/Hitachi DSCM microdrive in a PXA PCMCIA slot - not in "Akita"
- On-chip OHCI USB controller
- On-chip LCD controller
- On-chip Real Time Clock
- TI ADS7846 touchscreen controller on SSP bus
- Maxim MAX1111 analog-digital converter on I^2C bus
- GPIO-connected keyboard controller and LEDs
- Secure Digital card connected to PXA MMC/SD host
- Three on-chip UARTs

- WM8750 audio CODEC on I²C and I²S busses

The Palm Tungsten|E PDA (codename "Cheetah") emulation includes the following elements:

- Texas Instruments OMAP310 System-on-chip (ARM 925T core)
- ROM and RAM memories (ROM firmware image can be loaded with -option-rom)
- On-chip LCD controller
- On-chip Real Time Clock
- TI TSC2102i touchscreen controller / analog-digital converter / Audio CODEC, connected through MicroWire and I²S busses
- GPIO-connected matrix keypad
- Secure Digital card connected to OMAP MMC/SD host
- Three on-chip UARTs

Nokia N800 and N810 internet tablets (known also as RX-34 and RX-44 / 48) emulation supports the following elements:

- Texas Instruments OMAP2420 System-on-chip (ARM 1136 core)
- RAM and non-volatile OneNAND Flash memories
- Display connected to EPSON remote framebuffer chip and OMAP on-chip display controller and a LS041y3 MIPI DBI-C controller
- TI TSC2301 (in N800) and TI TSC2005 (in N810) touchscreen controllers driven through SPI bus
- National Semiconductor LM8323-controlled qwerty keyboard driven through I²C bus
- Secure Digital card connected to OMAP MMC/SD host
- Three OMAP on-chip UARTs and on-chip STI debugging console
- A Bluetooth(R) transceiver and HCI connected to an UART
- Mentor Graphics "Inventra" dual-role USB controller embedded in a TI TUSB6010 chip - only USB host mode is supported
- TI TMP105 temperature sensor driven through I²C bus
- TI TWL92230C power management companion with an RTC on I²C bus
- Nokia RETU and TAHVO multi-purpose chips with an RTC, connected through CBUS

The Luminary Micro Stellaris LM3S811EVB emulation includes the following devices:

- Cortex-M3 CPU core.
- 64k Flash and 8k SRAM.
- Timers, UARTs, ADC and I²C interface.
- OSRAM Pictiva 96x16 OLED with SSD0303 controller on I²C bus.

The Luminary Micro Stellaris LM3S6965EVB emulation includes the following devices:

- Cortex-M3 CPU core.
- 256k Flash and 64k SRAM.
- Timers, UARTs, ADC, I²C and SSI interfaces.
- OSRAM Pictiva 128x64 OLED with SSD0323 controller connected via SSI.

The ST STM32F429I-Discovery board:
- STM32F429ZI (Cortex-M4 CPU core, FP not emulated).
- 2048k Flash and 192k SRAM.
- 2 LEDs, active high (Green PG13, Red PG14).

The ST STM32F4-Discovery board:
- STM32F407VG (Cortex-M4 CPU core, FP not emulated).
- 1024k Flash and 128k SRAM.
- 4 LEDs, active high (Green PD12, Orange PD13, Red PD14, Blue PD15).

The ST NUCLEO-F103RB board:
- STM32F103RB (Cortex-M3 CPU core).
- 128k Flash and 20k SRAM.
- 1 LED, active high (Green PA5).

The NetduinoPlus2 board:
- STM32F405RG (Cortex-M4 CPU core, FP not emulated).
- 1024k Flash and 128k SRAM.
- 1 LED, active high (Green PA10).

The NetduinoGo board:
- STM32F405RG (Cortex-M4 CPU core, FP not emulated).
- 1024k Flash and 128k SRAM.
- 8 LEDs, active high (White PB4-PB9, PC4-PC9).

The LeafLab Maple board:
- STM32F103RB (Cortex-M3 CPU core).
- 128k Flash and 20k SRAM.
- 1 LED, active high (Green PA5).

The Olimex STM32-H103 board:
- STM32F103RB (Cortex-M3 CPU core).
- 128k Flash and 20k SRAM.
- 1 LED, active low (Green PC12).

The Olimex STM32-P103 board:
- STM32F103RB (Cortex-M3 CPU core).
- 128k Flash and 20k SRAM.
- 1 LED, active low (Red PC12).

The Olimex OLIMEXINO-STM32 board:
- STM32F103RB (Cortex-M3 CPU core).
- 128k Flash and 20k SRAM.
- 2 LEDs, active high (Green PA5, Yellow PA1).

The Olimex STM32-P107 board:

- STM32F107VC (Cortex-M3 CPU core).
- 256k Flash and 64k SRAM.
- 2 LEDs, active high (Green PC6, Yellow PC7).

The Olimex STM32-E407 board:

- STM32F407ZG (Cortex-M4 CPU core, FP not emulated).
- 1024k Flash and 128k SRAM.
- 1 LED, active low (Green PC13).

The Freecom MusicPal internet radio emulation includes the following elements:

- Marvell MV88W8618 ARM core.
- 32 MB RAM, 256 KB SRAM, 8 MB flash.
- Up to 2 16550 UARTs
- MV88W8xx8 Ethernet controller
- MV88W8618 audio controller, WM8750 CODEC and mixer
- 128×64 display with brightness control
- 2 buttons, 2 navigation wheels with button function

The Siemens SX1 models v1 and v2 (default) basic emulation. The emulation includes the following elements:

- Texas Instruments OMAP310 System-on-chip (ARM 925T core)
- ROM and RAM memories (ROM firmware image can be loaded with -pflash) V1 1 Flash of 16MB and 1 Flash of 8MB V2 1 Flash of 32MB
- On-chip LCD controller
- On-chip Real Time Clock
- Secure Digital card connected to OMAP MMC/SD host
- Three on-chip UARTs

A Linux 2.6 test image is available on the QEMU web site. More information is available in the QEMU mailing-list archive.

The following options are specific to the ARM emulation:

`-semihosting`

Enable semihosting syscall emulation.

On ARM this implements the "Angel" interface.

Note that this allows guest direct access to the host filesystem, so should only be used with trusted guest OS.

3.6 ColdFire System emulator

Use the executable `qemu-system-m68k` to simulate a ColdFire machine. The emulator is able to boot a uClinux kernel.

The M5208EVB emulation includes the following devices:

- MCF5208 ColdFire V2 Microprocessor (ISA A+ with EMAC).
- Three Two on-chip UARTs.
- Fast Ethernet Controller (FEC)

The AN5206 emulation includes the following devices:

- MCF5206 ColdFire V2 Microprocessor.
- Two on-chip UARTs.

The following options are specific to the ColdFire emulation:

`-semihosting`
: Enable semihosting syscall emulation.

  On M68K this implements the "ColdFire GDB" interface used by libgloss.

  Note that this allows guest direct access to the host filesystem, so should only be used with trusted guest OS.

3.7 Cris System emulator

TODO

3.8 Microblaze System emulator

TODO

3.9 SH4 System emulator

TODO

3.10 Xtensa System emulator

Two executables cover simulation of both Xtensa endian options, `qemu-system-xtensa` and `qemu-system-xtensaeb`. Two different machine types are emulated:

- Xtensa emulator pseudo board "sim"
- Avnet LX60/LX110/LX200 board

The sim pseudo board emulation provides an environment similar to one provided by the proprietary Tensilica ISS. It supports:

- A range of Xtensa CPUs, default is the DC232B
- Console and filesystem access via semihosting calls

The Avnet LX60/LX110/LX200 emulation supports:

- A range of Xtensa CPUs, default is the DC232B
- 16550 UART

– OpenCores 10/100 Mbps Ethernet MAC

The following options are specific to the Xtensa emulation:

`-semihosting`

Enable semihosting syscall emulation.

Xtensa semihosting provides basic file IO calls, such as open/read/write/seek/select. Tensilica baremetal libc for ISS and linux platform "sim" use this interface.

Note that this allows guest direct access to the host filesystem, so should only be used with trusted guest OS.

4 QEMU User space emulator

4.1 Supported Operating Systems

The following OS are supported in user space emulation:

- Linux (referred as qemu-linux-user)
- BSD (referred as qemu-bsd-user)

4.2 Features

QEMU user space emulation has the following notable features:

System call translation:
QEMU includes a generic system call translator. This means that the parameters of the system calls can be converted to fix endianness and 32/64-bit mismatches between hosts and targets. IOCTLs can be converted too.

POSIX signal handling:
QEMU can redirect to the running program all signals coming from the host (such as `SIGALRM`), as well as synthesize signals from virtual CPU exceptions (for example `SIGFPE` when the program executes a division by zero).

QEMU relies on the host kernel to emulate most signal system calls, for example to emulate the signal mask. On Linux, QEMU supports both normal and real-time signals.

Threading:
On Linux, QEMU can emulate the `clone` syscall and create a real host thread (with a separate virtual CPU) for each emulated thread. Note that not all targets currently emulate atomic operations correctly. x86 and ARM use a global lock in order to preserve their semantics.

QEMU was conceived so that ultimately it can emulate itself. Although it is not very useful, it is an important test to show the power of the emulator.

4.3 Linux User space emulator

4.3.1 Quick Start

In order to launch a Linux process, QEMU needs the process executable itself and all the target (x86) dynamic libraries used by it.

- On x86, you can just try to launch any process by using the native libraries:

  ```
  qemu-i386 -L / /bin/ls
  ```

  `-L /` tells that the x86 dynamic linker must be searched with a `/` prefix.

- Since QEMU is also a linux process, you can launch QEMU with QEMU (NOTE: you can only do that if you compiled QEMU from the sources):

  ```
  qemu-i386 -L / qemu-i386 -L / /bin/ls
  ```

- On non x86 CPUs, you need first to download at least an x86 glibc (`qemu-runtime-i386-XXX-.tar.gz` on the QEMU web page). Ensure that `LD_LIBRARY_PATH` is not set:

  ```
  unset LD_LIBRARY_PATH
  ```

  Then you can launch the precompiled `ls` x86 executable:

  ```
  qemu-i386 tests/i386/ls
  ```

  You can look at `scripts/qemu-binfmt-conf.sh` so that QEMU is automatically launched by the Linux kernel when you try to launch x86 executables. It requires the `binfmt_misc` module in the Linux kernel.
- The x86 version of QEMU is also included. You can try weird things such as:

  ```
  qemu-i386 /usr/local/qemu-i386/bin/qemu-i386 \
            /usr/local/qemu-i386/bin/ls-i386
  ```

4.3.2 Wine launch

- Ensure that you have a working QEMU with the x86 glibc distribution (see previous section). In order to verify it, you must be able to do:

  ```
  qemu-i386 /usr/local/qemu-i386/bin/ls-i386
  ```
- Download the binary x86 Wine install (`qemu-XXX-i386-wine.tar.gz` on the QEMU web page).
- Configure Wine on your account. Look at the provided script `/usr/local/qemu-i386/bin/wine-conf.sh`. Your previous `${HOME}/.wine` directory is saved to `${HOME}/.wine.org`.
- Then you can try the example `putty.exe`:

  ```
  qemu-i386 /usr/local/qemu-i386/wine/bin/wine \
            /usr/local/qemu-i386/wine/c/Program\ Files/putty.exe
  ```

4.3.3 Command line options

`qemu-i386 [-h] [-d] [-L` *`path`*`] [-s` *`size`*`] [-cpu` *`model`*`] [-g` *`port`*`] [-B` *`offset`*`] [-R` *`size`*`] pro-`
`gram` `[`*`arguments...`*`]`

`-h` Print the help

`-L path` Set the x86 elf interpreter prefix (default=/usr/local/qemu-i386)

`-s size` Set the x86 stack size in bytes (default=524288)

`-cpu model`
Select CPU model (-cpu help for list and additional feature selection)

`-E` *`var=value`*
Set environment *var* to *value*.

`-U` *`var`* Remove *var* from the environment.

`-B offset` Offset guest address by the specified number of bytes. This is useful when the address region required by guest applications is reserved on the host. This option is currently only supported on some hosts.

`-R size` Pre-allocate a guest virtual address space of the given size (in bytes). "G", "M", and "k" suffixes may be used when specifying the size.

Debug options:

`-d item1,...`
Activate logging of the specified items (use '-d help' for a list of log items)

`-p pagesize`
Act as if the host page size was 'pagesize' bytes

`-g port` Wait gdb connection to port

`-singlestep`
Run the emulation in single step mode.

Environment variables:

`QEMU_STRACE`
Print system calls and arguments similar to the 'strace' program (NOTE: the actual 'strace' program will not work because the user space emulator hasn't implemented ptrace). At the moment this is incomplete. All system calls that don't have a specific argument format are printed with information for six arguments. Many flag-style arguments don't have decoders and will show up as numbers.

4.3.4 Other binaries

`qemu-alpha` TODO.

`qemu-armeb` TODO.

`qemu-arm` is also capable of running ARM "Angel" semihosted ELF binaries (as implemented by the arm-elf and arm-eabi Newlib/GDB configurations), and arm-uclinux bFLT format binaries.

`qemu-m68k` is capable of running semihosted binaries using the BDM (m5xxx-ram-hosted.ld) or m68k-sim (sim.ld) syscall interfaces, and coldfire uClinux bFLT format binaries.

The binary format is detected automatically.

`qemu-cris` TODO.

`qemu-i386` TODO. `qemu-x86_64` TODO.

`qemu-microblaze` TODO.

`qemu-mips` TODO. `qemu-mipsel` TODO.

`qemu-ppc64abi32` TODO. `qemu-ppc64` TODO. `qemu-ppc` TODO.

`qemu-sh4eb` TODO. `qemu-sh4` TODO.

`qemu-sparc` can execute Sparc32 binaries (Sparc32 CPU, 32 bit ABI).

`qemu-sparc32plus` can execute Sparc32 and SPARC32PLUS binaries (Sparc64 CPU, 32 bit ABI).

`qemu-sparc64` can execute some Sparc64 (Sparc64 CPU, 64 bit ABI) and SPARC32PLUS binaries (Sparc64 CPU, 32 bit ABI).

4.4 BSD User space emulator

4.4.1 BSD Status

– target Sparc64 on Sparc64: Some trivial programs work.

4.4.2 Quick Start

In order to launch a BSD process, QEMU needs the process executable itself and all the target dynamic libraries used by it.

- On Sparc64, you can just try to launch any process by using the native libraries:

  `qemu-sparc64 /bin/ls`

4.4.3 Command line options

`qemu-sparc64 [-h] [-d] [-L` *`path`*`] [-s` *`size`*`] [-bsd` *`type`*`]` *`program`* `[`*`arguments...`*`]`

`-h` Print the help

`-L path` Set the library root path (default=/)

`-s size` Set the stack size in bytes (default=524288)

`-ignore-environment`
Start with an empty environment. Without this option, the initial environment is a copy of the caller's environment.

`-E` *`var`*`=`*`value`*
Set environment *var* to *value*.

`-U` *`var`* Remove *var* from the environment.

`-bsd type` Set the type of the emulated BSD Operating system. Valid values are FreeBSD, NetBSD and OpenBSD (default).

Debug options:

`-d item1,...`
Activate logging of the specified items (use '-d help' for a list of log items)

`-p pagesize`
Act as if the host page size was 'pagesize' bytes

`-singlestep`
Run the emulation in single step mode.

Appendix A Implementation notes

A.1 CPU emulation

A.1.1 x86 and x86-64 emulation

QEMU x86 target features:

- The virtual x86 CPU supports 16 bit and 32 bit addressing with segmentation. LDT/GDT and IDT are emulated. VM86 mode is also supported to run DOSEMU. There is some support for MMX/3DNow!, SSE, SSE2, SSE3, SSSE3, and SSE4 as well as x86-64 SVM.
- Support of host page sizes bigger than 4KB in user mode emulation.
- QEMU can emulate itself on x86.
- An extensive Linux x86 CPU test program is included `tests/test-i386`. It can be used to test other x86 virtual CPUs.

Current QEMU limitations:

- Limited x86-64 support.
- IPC syscalls are missing.
- The x86 segment limits and access rights are not tested at every memory access (yet). Hopefully, very few OSes seem to rely on that for normal use.

A.1.2 ARM emulation

- Full ARM 7 user emulation.
- NWFPE FPU support included in user Linux emulation.
- Can run most ARM Linux binaries.

A.1.3 MIPS emulation

- The system emulation allows full MIPS32/MIPS64 Release 2 emulation, including privileged instructions, FPU and MMU, in both little and big endian modes.
- The Linux userland emulation can run many 32 bit MIPS Linux binaries.

Current QEMU limitations:

- Self-modifying code is not always handled correctly.
- 64 bit userland emulation is not implemented.
- The system emulation is not complete enough to run real firmware.
- The watchpoint debug facility is not implemented.

A.1.4 PowerPC emulation

- Full PowerPC 32 bit emulation, including privileged instructions, FPU and MMU.
- Can run most PowerPC Linux binaries.

A.1.5 Sparc32 and Sparc64 emulation

- Full SPARC V8 emulation, including privileged instructions, FPU and MMU. SPARC V9 emulation includes most privileged and VIS instructions, FPU and I/D MMU. Alignment is fully enforced.
- Can run most 32-bit SPARC Linux binaries, SPARC32PLUS Linux binaries and some 64-bit SPARC Linux binaries.

Current QEMU limitations:

- IPC syscalls are missing.
- Floating point exception support is buggy.
- Atomic instructions are not correctly implemented.
- There are still some problems with Sparc64 emulators.

A.1.6 Xtensa emulation

- Core Xtensa ISA emulation, including most options: code density, loop, extended L32R, 16- and 32-bit multiplication, 32-bit division, MAC16, miscellaneous operations, boolean, FP coprocessor, coprocessor context, debug, multiprocessor synchronization, conditional store, exceptions, relocatable vectors, unaligned exception, interrupts (including high priority and timer), hardware alignment, region protection, region translation, MMU, windowed registers, thread pointer, processor ID.
- Not implemented options: data/instruction cache (including cache prefetch and locking), XLMI, processor interface. Also options not covered by the core ISA (e.g. FLIX, wide branches) are not implemented.
- Can run most Xtensa Linux binaries.
- New core configuration that requires no additional instructions may be created from overlay with minimal amount of hand-written code.

A.2 Translator Internals

QEMU is a dynamic translator. When it first encounters a piece of code, it converts it to the host instruction set. Usually dynamic translators are very complicated and highly CPU dependent. QEMU uses some tricks which make it relatively easily portable and simple while achieving good performances.

QEMU's dynamic translation backend is called TCG, for "Tiny Code Generator". For more information, please take a look at `tcg/README`.

Some notable features of QEMU's dynamic translator are:

CPU state optimisations:

The target CPUs have many internal states which change the way it evaluates instructions. In order to achieve a good speed, the translation phase considers that some state information of the virtual CPU cannot change in it. The state is recorded in the Translation Block (TB). If the state changes (e.g. privilege level), a new TB will be generated and the previous TB won't be used anymore until the state matches the state recorded in the previous TB. The same idea can be applied to other aspects of the CPU state. For example, on x86, if the

SS, DS and ES segments have a zero base, then the translator does not even generate an addition for the segment base.

Direct block chaining:

After each translated basic block is executed, QEMU uses the simulated Program Counter (PC) and other cpu state information (such as the CS segment base value) to find the next basic block.

In order to accelerate the most common cases where the new simulated PC is known, QEMU can patch a basic block so that it jumps directly to the next one.

The most portable code uses an indirect jump. An indirect jump makes it easier to make the jump target modification atomic. On some host architectures (such as x86 or PowerPC), the `JUMP` opcode is directly patched so that the block chaining has no overhead.

Self-modifying code and translated code invalidation:

Self-modifying code is a special challenge in x86 emulation because no instruction cache invalidation is signaled by the application when code is modified.

User-mode emulation marks a host page as write-protected (if it is not already read-only) every time translated code is generated for a basic block. Then, if a write access is done to the page, Linux raises a SEGV signal. QEMU then invalidates all the translated code in the page and enables write accesses to the page. For system emulation, write protection is achieved through the software MMU.

Correct translated code invalidation is done efficiently by maintaining a linked list of every translated block contained in a given page. Other linked lists are also maintained to undo direct block chaining.

On RISC targets, correctly written software uses memory barriers and cache flushes, so some of the protection above would not be necessary. However, QEMU still requires that the generated code always matches the target instructions in memory in order to handle exceptions correctly.

Exception support:

longjmp() is used when an exception such as division by zero is encountered.

The host SIGSEGV and SIGBUS signal handlers are used to get invalid memory accesses. QEMU keeps a map from host program counter to target program counter, and looks up where the exception happened based on the host program counter at the exception point.

On some targets, some bits of the virtual CPU's state are not flushed to the memory until the end of the translation block. This is done for internal emulation state that is rarely accessed directly by the program and/or changes very often throughout the execution of a translation block—this includes condition codes on x86, delay slots on SPARC, conditional execution on ARM, and so on. This state is stored for each target instruction, and looked up on exceptions.

MMU emulation:

For system emulation QEMU uses a software MMU. In that mode, the MMU virtual to physical address translation is done at every memory access.

QEMU uses an address translation cache (TLB) to speed up the translation. In order to avoid flushing the translated code each time the MMU mappings change, all caches in QEMU are physically indexed. This means that each basic block is indexed with its physical address.

In order to avoid invalidating the basic block chain when MMU mappings change, chaining is only performed when the destination of the jump shares a page with the basic block that is performing the jump.

The MMU can also distinguish RAM and ROM memory areas from MMIO memory areas. Access is faster for RAM and ROM because the translation cache also hosts the offset between guest address and host memory. Accessing MMIO memory areas instead calls out to C code for device emulation. Finally, the MMU helps tracking dirty pages and pages pointed to by translation blocks.

A.3 QEMU compared to other emulators

Like bochs [1], QEMU emulates an x86 CPU. But QEMU is much faster than bochs as it uses dynamic compilation. Bochs is closely tied to x86 PC emulation while QEMU can emulate several processors.

Like Valgrind [2], QEMU does user space emulation and dynamic translation. Valgrind is mainly a memory debugger while QEMU has no support for it (QEMU could be used to detect out of bound memory accesses as Valgrind, but it has no support to track uninitialised data as Valgrind does). The Valgrind dynamic translator generates better code than QEMU (in particular it does register allocation) but it is closely tied to an x86 host and target and has no support for precise exceptions and system emulation.

EM86 [3] is the closest project to user space QEMU (and QEMU still uses some of its code, in particular the ELF file loader). EM86 was limited to an alpha host and used a proprietary and slow interpreter (the interpreter part of the FX!32 Digital Win32 code translator [4]).

TWIN from Willows Software was a Windows API emulator like Wine. It is less accurate than Wine but includes a protected mode x86 interpreter to launch x86 Windows executables. Such an approach has greater potential because most of the Windows API is executed natively but it is far more difficult to develop because all the data structures and function parameters exchanged between the API and the x86 code must be converted.

User mode Linux [5] was the only solution before QEMU to launch a Linux kernel as a process while not needing any host kernel patches. However, user mode Linux requires heavy kernel patches while QEMU accepts unpatched Linux kernels. The price to pay is that QEMU is slower.

The Plex86 [6] PC virtualizer is done in the same spirit as the now obsolete qemu-fast system emulator. It requires a patched Linux kernel to work (you cannot launch the same kernel on your PC), but the patches are really small. As it is a PC virtualizer (no emulation is done except for some privileged instructions), it has the potential of being faster than QEMU. The downside is that a complicated (and potentially unsafe) host kernel patch is needed.

The commercial PC Virtualizers (VMWare [7], VirtualPC [8]) are faster than QEMU (without virtualization), but they all need specific, proprietary and potentially unsafe host drivers. Moreover, they are unable to provide cycle exact simulation as an emulator can.

VirtualBox [9], Xen [10] and KVM [11] are based on QEMU. QEMU-SystemC [12] uses QEMU to simulate a system where some hardware devices are developed in SystemC.

A.4 Bibliography

[1] `http://bochs.sourceforge.net/`, the Bochs IA-32 Emulator Project, by Kevin Lawton et al.

[2] `http://www.valgrind.org/`, Valgrind, an open-source memory debugger for GNU/Linux.

[3] `http://ftp.dreamtime.org/pub/linux/Linux-Alpha/em86/v0.2/docs/em86.html`, the EM86 x86 emulator on Alpha-Linux.

[4] `http://www.usenix.org/publications/library/proceedings/usenix-nt97/full_papers/chernoff/chernoff.pdf`, DIGITAL FX!32: Running 32-Bit x86 Applications on Alpha NT, by Anton Chernoff and Ray Hookway.

[5] `http://user-mode-linux.sourceforge.net/`, The User-mode Linux Kernel.

[6] `http://www.plex86.org/`, The new Plex86 project.

[7] `http://www.vmware.com/`, The VMWare PC virtualizer.

[8] `https://www.microsoft.com/download/details.aspx?id=3702`, The VirtualPC PC virtualizer.

[9] `http://virtualbox.org/`, The VirtualBox PC virtualizer.

[10] `http://www.xen.org/`, The Xen hypervisor.

[11] `http://www.linux-kvm.org/`, Kernel Based Virtual Machine (KVM).

[12] `http://www.greensocs.com/projects/QEMUSystemC`, QEMU-SystemC, a hardware co-simulator.

Appendix B License

QEMU is a trademark of Fabrice Bellard.

QEMU is released under the GNU General Public License (TODO: add link). Parts of QEMU have specific licenses, see file LICENSE.

TODO (refer to file LICENSE, include it, include the GPL?)

Appendix C Index

C.1 Concept Index

This is the main index. Should we combine all keywords in one index? TODO

O

Q

S

U

C.2 Function Index

This index could be used for command line options and monitor functions.

—

A

B

C

D

E

G

H

I

J

K

L

M

N

C.3 Keystroke Index

This is a list of all keystrokes which have a special function in system emulation.

C.4 Program Index

(Index is nonexistent)

C.5 Data Type Index

This index could be used for qdev device names and options.

(Index is nonexistent)

C.6 Variable Index

(Index is nonexistent)