virt-lib Quick Reference

virtlib stores its configuration for each *domain* (virtual) in /etc/libvirt/qemu*.xml. You should not manually modify anything in this directory, but you can safely do any read-only operation. To edit/export/whatever, the *virsh* command has several options that will help you out.

The xml files are, well, xml. The format is documented at https://libvirt.org/formatdomain.html which is guaranteed to either help you, or put you to sleep.

virt-top

This is not always installed by default. When run, it will bring up a full console window that shows all virtuals running on a system, similar to the top command.

virsh

The main controller for virt-lib. The basic format of the command is

```
virsh command parameters
```

. *man virsh* or simply *virsh –help* are your friends.

Some common commands are:

- **list –all** - list all domains virt-lib knows about and show their state (running, off, whatever)
- **vncdisplay *name*** - show vnc display number (not port) assigned
- **dommemstat *name*** - show allocated RAM
- **domstats *name* | grep vcpu.current** - displays number of virtual cpu's assigned
- **domblklist *name*** - show attached block devices
- **domiflist *name*** - list all domain interfaces
- **start *name*** - starts the domain named **name** (name from list –all)
- **reboot *name***
- **shutdown *name***
- **dumpxml *name*** dumps the config to STDOUT
- **list –autostart** - list all domains which will autostart
- **autostart domain** - set domain to be autostarted in the future
- **autostart –disable domain** - unset autostart flag for domain
- **console domain** - attaches to the serial console of domain assuming the port has been set up.
- Attach/Detach network interfaces (https://computingforgeeks.com/managing-kvm-network-interfaces-in-linux/)
  - domiflist *domain*
  - attach-interface *domain* bridge *nameonhv* –config –target vnet3 –mac 00:16:3e:xx:xx:xx –model virtio
    - This will attach a new bridge to domain *domain*, using *naeonhv* defined on hypervisor, called vnet3 on the virtual, with the mac address set to whatever you

use.
    - detach-interface *domain* –type bridge –mac 00:16:3e:xx:xx:xx –config
- Attach/Detach Block Device
    - `virsh detach-disk router sda --config`
    - `virsh attach-disk router /dev/vg/lvname sda --config --targetbus=virtio`
- Remove virtual image (config file only)

```
virsh undefine domainname
```

- **change-media *name drive*** - Insert or Eject a CDROM
    - `virsh change-media domainname driveletter --eject`
    - `virsh change-media domainname driveletter /path/to/image/ --insert`
- Rename Domain
    - `virsh shutdown domainname ; virsh domrename domainname newname ; virsh start newname`

Boot from CD ROM

This is actually not intuitive. I'm going to describe how to do this from a pretty complex setup with no GUI; adjust as needed. In this case, we need to boot the virtual myvirt from a gparted cdrom image located in /media/isos/gparted-live-1.3.0-1-amd64.iso so we can modify the partitions. However, we did not have a cdrom image for the installation, so one has not been created.

Both assume the virtual has been turned off

```
virsh shutdown myvirt
```

Some people do not want to manually edit the XML used to configure the virtuals, others prefer it. Some things, like setting a boot menu with a realistic timeout appear to require it; I have not found a way to do it through the virsh command.

I use the –config flag a lot. If you use this flag, any changes you make are written to the config file and will persist across boots.

without manually editing configuration

1. First, see if there is a cdrom installed. One simple way is to dump the xml and grep for cdrom

```
virsh dumpxml myvirt | grep cdrom
```

2. Once this is done and you have a cdrom, you should be able to tell which one it is with

```
virsh domblklist myvirt
```

   which shows the block devices attached to myvirt.

3. Mount your image on the CDROM
   1. If you have one and you know which drive it is

```
virsh change-media dd-113-virt hdb /media/xen-store/isos/gparted-live-1.3.0-1-amd64.iso --insert --config
```

      This is assuming the drive was hdb

   2. If you do not have a CDROM, use

```
virsh attach-disk myvirt /media/xen-store/isos/gparted-live-1.3.0-1-amd64.iso hdb --driver file --type cdrom --mode readonly --config
```

      This assumes hdb did not show up as used in your domblklist above.

I'm not afraid to edit the config

1. Edit the config file with

```
virsh edit myvirt
```

2. Look for something like *<disk type='file' device='cdrom'>*
3. If it exists, add

```
<source file='/media/isos/gparted-live-1.3.0-1-amd64.iso'/>
```

4. Look for the section *<os>*, generally near the top and add lines if necessary
   1. `<bootmenu enable='yes' timeout='5000'/>`
   2. This will enable the boot menu, and have a wait of 5 seconds (5000 milliseconds) for you to choose.
5. If you want, change the boot order by adding

```
<boot dev='cdrom'/>
```

   above the boot entry if it is not already set.

Starting the virtual

1. Start the virtual with

```
virsh start myvirt
```

2. Immediately make the VNC connection (if you followed the manual edit, you have 5 seconds)
3. When prompted, press ESC to choose the CDROM drive (assuming you didn't set it as the default)

Cleanup

Do the following if you want to remove the CDROM when you're done. Note: this is likely required if you set the CDROM as the primary boot device.

```
virsh change-media myvirt hdb --eject --config
```

Remove a network from the entire system

```
virsh net-list --all
virsh net-destroy br1
virsh net-undefine br1
virsh net-list --all
```

convert config file to native

```
virsh -c xen:/// domxml-to-native --format xen-xl \
    /path/to/libvirt/vm.domxml.cfg > vm.xenxl.cfg
```

New Install of Windows with virtio

virt-install does not allow you to use two ISO's as cdrom images by default. This can be a problem when you want to do a new install, but want to use virtio on Windows (which requires a second ISO. You may be able to do a change-media, but the following, taken from https://superuser.com/questions/147419/using-virt-install-to-mount-multiple-cdrom-drives-images will do the trick with only one extra step.

Note: You may have to manually start the virtual several times during the installation. Windows reboots 2-3 times during an installation, and sometimes this is not handled well by virsh.

1. Add the –print-xml flag to virt-install, and pipe the output to a file.
2. Manually edit the resulting xml file, adding a second cdrom drive (copy/paste)
    1. Change the drive letter
    2. Change the target to point to virtio-win.iso
3. run

    ```
    virsh create name_of_xml_file
    ```

    to begin the installation

4. After installation is complete, install the remainder of the virtio drivers for maximum efficiency
5. To permanently define after installation
    1. Edit XML and remove the second CDROM (if desired)
    2. `virsh define name_of_xml_file`

Upgrading a disk to virtio (Unix)

I messed up and did not use virtio as the bus for one of my setups, which resulted in horrible disk I/O. I could have rebuilt the virtual (it was a simple install and I had not gone very far), but I decided to learn how to do it the "right way". The trick here is to detach the disk, then re-attach it with the proper parameters.

Warning Linux (and FreeBSD) have the virtio drivers built in, but Microsoft products do not. Read the section on Microsoft products if you are trying to do this with Windows.

In my case, I wanted to reconfigure the boot drive, so I needed the machine down. It was running under the scsi bus, and I wanted virtio. Note that his is using LVM2 as the back end.

```
# get a list of all the block devices
virsh domblklist router
# detach the first drive (sda) from the domain router
virsh detach-disk router sda --config
# reattach it using the virtio bus.
virsh attach-disk router /dev/vg/lvname sda --config --targetbus=virtio
```

Normally, you attach/detach disks from a running system. The –config parameter allows you to do it on an inactive domain; it just erases the entry, then rebuilds it.

Upgrading a disk to virtio (Windows)

For Windows, you need to have the correct device drivers installed before you change the disk. This assumes the domain is running

1. Get a copy of the Windows VirtIO-win in iso onto your machine, someplace virsh can access, and mount the ISO
    1. Download ISO

        ```
        cd /media/virtstorage && wget
        https://fedorapeople.org/groups/virt/virtio-win/direct-
        downloads/stable-virtio/virtio-win.iso
        ```

    2. Mount it on a running system

        ```
        virsh attach-disk domainname /media/virtstorage/stable-
        virtio/virtio-win.iso hdc --type cdrom --mode readonly
        ```

2. Create a temporary disk and attach it. Use virtio for the format of it.

1. `virsh attach-disk domainname /path/to/disk/image sdb --targetbus=virtio`

3. Open your console to the windows domain
    1. Find the new diskdisk (sdb). It will have a yellow triangle, indicating it can not be read because there is no device driver. Right click and say “Update Driver”, then go find the drivers from the CD.
    2. Shut down your Windows guest
4. Detach, then attach the primary drive.

```
# get a list of all the block devices
virsh domblklist router
# detach the first drive (sda) from the domain router
virsh detach-disk domainname sda --config
# reattach it using the virtio bus.
virsh attach-disk domainname /dev/vg/lvname sda --config --targetbus=virtio
```

5. Start the Windows guest back up. Since the drivers are already installed, it should come back up.

I did this with a Windows 10 installation on my workstation. Prior to using virtio, it would take, literally, 5-7 minutes after boot before I could do anything, and it was very sluggish after that. Once I used virtio, it was almost bare hardware speeds.

Shutdown and restart of Windows guests

Problems shutting down with //virsh//

After installing the win-virtio package, you should see QEMU Guest Agent running as a service. With this running, you can use qemu-guest-agent to manage shutdown and reboot. This is much more reliable than using ACPI.

Important Shut down the virtual before doing the following.

Edit the guest

```
virsh edit DOMAIN
```

Place the following block under the <devices> section. I usually put it after the <console> sections, but just so long as it is in the <devices> section of the config. Virsh will rearrainge it for you anyway.

```
<channel type="unix">
  <source mode="bind"/>
  <target type="virtio" name="org.qemu.guest_agent.0"/>
</channel>
```

Start the virtual back up. Once that is done you can use the following commands much more reliably.

```
virsh shutdown DOMAIN
virsh reboot DOMAIN
```

This also allows you to execute *qemu-agent-command* from within *virsh*, but this is strongly discouraged unless you have researched what is going on. Any changes you make can cause instability in libvirt, since they are bypassing virsh. Kind of like using hdparm, where you can do Really Bad Things to your hard disk if you don't know for sure what you are doing. Do what you want, but be careful.

Using save/restore

An alternative is to use *virsh save* to shut down and *virsh restore* to recover. In this case, use

```
virsh save domainname /path/to/save/image
```

to save the image. It will suspend the virtual, then save processor/memory/whatever to the file /path/to/save/image.

Once this is done, you can do whatever you needed to do, then use the following command to restore it.

```
virsh restore /path/to/save/image
```

Windows servers will not restart

I'm having a problem with Windows virtuals not rebooting. When you issue the restart command, they shut off and don't come back up. As a band aid, I have a script running on the hypervisor with a cron job, every 5 minutes.

This script has been tested on our machines, but I'm sure there are some issues with it. Just do a

```
virsh list --all
```

and select the domains you want to ensure are running all the time. Place them in the array that has DOMAIN1 and DOMAIN2 (ie, replace DOMAIN1 with your first choice, etc...).

When called, checkVirtuals will look for each of the domains and see if they are running (using *virsh list*). If they are not running, it will place a flag file in /tmp/DOMAIN.down. The next time it is run, it will note the domain is still not down, and the flag file exists, so it will start the domain (using *virsh start DOMAIN* and delete the flag file.

I call this every 5 minutes from cron, thus, the max downtime will be 10 minutes, with an average of 5.

WARNING: Remember this is running. If you need to take a virtual down for some reason, as long as this script is running, it will blindly go ahead and restart it.

checkVirtuals

```
#! /usr/bin/env perl

use strict;
use warnings;

my @servers = (
   'DOMAIN1',
   'DOMAIN2'
   );

my $virsh = '/usr/bin/virsh start ';

my $output = `virsh list`;

foreach my $server ( @servers ) {
   if ( $output =~ m/$server/ ) {
      unlink "/tmp/$server.down" if  -e "/tmp/$server.down";
   } else {
      if ( -e "/tmp/$server.down" ) {
         print "$server has been down for a while, starting back up\n";
         `$virsh $server`;
         unlink "/tmp/$server.down";
      } else {
         `touch /tmp/$server.down`;
      }
   }
}

1;
```

Replacing Network Interfaces

Sometimes you need to undefine and redefine a network interface. This is actually fairly simple to do.

```
# get a list of all network interfaces in domain
virsh domiflist domain
# remove the one you want. Use values (type, mac) from above command
virsh detach-interface domain --type bridge --mac ##:##:##:##:##:## --config
# redefine it. Use values from above command, or change as needed
virsh attach-interface domain --type bridge --model virtio --source
br_private --mac ##:##:##:##:##:## -config
```

In this example, *domain* is the name of the domain to be worked on. I use domiflist to get information about the network interfaces as I need the type and mac. The –config makes it permanent, writing it to the config file.

In my case, I had built a Windows domain without using virtio, and I wanted to change it. This was

actually the simplest way I found to do it.

Naming Network Links

to give a domain interface a static name when it is run, you can add the following to the interface definition.

```
<target dev='dom113'/>
```

NOTE: the name must not begin with 'vnet', 'vif', 'macvtap', or 'macvlan', which are prefixes reserved by libvirt and certain hypervisors. See https://libvirt.org/formatdomain.html#overriding-the-target-element

```
   <interface type='bridge'>
      <mac address='00:16:3e:6b:f4:da'/>
      <source bridge='br_lan'/>
      <target dev='dom113'/>
      <model type='virtio'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x03'
function='0x0'/>
    </interface>
```

You can now do things like

```
ifconfig dom113
```

or

```
ip a show dom113
```

Adding USB Drive Image

Building and using a USB Thumbdrive image is a little weird, but it can be done. See Multiboot USB Thumb Drive for an example of one way I did it.

Links

- https://www.caretech.io/2017/10/18/converting-windows-vm-hard-disk-interface-to-virtio-with-proxmox-ve-5/
- https://docs.fedoraproject.org/en-US/quick-docs/creating-windows-virtual-machines-using-virtio-drivers/
- https://linuxhint.com/install_virtio_drivers_kvm_qemu_windows_vm/
- https://serverfault.com/questions/672253/how-to-configure-and-use-qemu-guest-agent-in-ubuntu-12-04-my-main-aim-is-to-get#691616
- https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/virtualization_a

dministration_guide/sect-qemu_guest_agent-running_the_qemu_guest_agent_on_a_windows_guest
- https://libvirt.org/formatdomain.html

From:
https://kb.unixservertech.com/ - **Unix Server Tech Knowledge Base**

Permanent link:
https://kb.unixservertech.com/unix/virtualization/virtlib/quickreference

Last update: **2026/04/15 00:17**