NAME

virt−inspector − Display operating system version and other information about a virtual machine

SYNOPSIS

```
virt−inspector [−−options] −d domname

virt−inspector [−−options] −a disk.img [−a disk.img ...]
```

Old-style:

```
virt−inspector domname

virt−inspector disk.img [disk.img ...]
```

DESCRIPTION

virt-inspector examines a virtual machine or disk image and tries to determine the version of the operating system and other information about the virtual machine.

Virt-inspector produces XML output for feeding into other programs.

In the normal usage, use `virt−inspector −d domname` where `domname` is the libvirt domain (see: `virsh list −−all`).

You can also run virt-inspector directly on disk images from a single virtual machine. Use `virt−inspector −a disk.img`. In rare cases a domain has several block devices, in which case you should list several *−a* options one after another, with the first corresponding to the guest's */dev/sda*, the second to the guest's */dev/sdb* and so on.

You can also run virt-inspector on install disks, live CDs, bootable USB keys and similar.

Virt-inspector can only inspect and report upon *one domain at a time*. To inspect several virtual machines, you have to run virt-inspector several times (for example, from a shell script for-loop).

Because virt-inspector needs direct access to guest images, it won't normally work over remote libvirt connections.

All of the information available from virt-inspector is also available through the core libguestfs inspection API (see ''INSPECTION'' in **guestfs** (3)). The same information can also be fetched using guestfish or via libguestfs bindings in many programming languages (see ''GETTING INSPECTION DATA FROM THE LIBGUESTFS API'').

OPTIONS

−−help
: Display brief help.

−a file
−−add file
: Add *file* which should be a disk image from a virtual machine. If the virtual machine has multiple block devices, you must supply all of them with separate *−a* options.

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

−a URI
−−add URI
: Add a remote disk. See ''ADDING REMOTE STORAGE'' in **guestfish** (1).

−−blocksize=512
−−blocksize=4096
−−blocksize
: This parameter sets the sector size of the disk image. It affects all explicitly added subsequent disks after this parameter. Using*−−bloc ksize* with no argument switches the disk sector size to the default value which is usually 512 bytes. See also ''guestfs_add_drive_opts'' in **guestfs** (3).

−c URI
−−connect URI
If using libvirt, connect to the given *URI*. If omitted, then we connect to the default libvirt hypervisor.

Libvirt is only used if you specify a `domname` on the command line. If you specify guest block devices directly (*−a*), then libvirt is not used at all.

−d guest
−−domain guest
Add all the disks from the named libvirt guest. Domain UUIDs can be used instead of names.

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

−−format=raw|qcow2|..
−−format
Specify the format of disk images given on the command line. If this is omitted then the format is autodetected from the content of the disk image.

If disk images are requested from libvirt, then this program asks libvirt for this information. In this case, the value of the format parameter is ignored.

If working with untrusted raw-format guest disk images, you should ensure the format is always specified.

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

−−key `ID`:key:KEY_STRING
Use the specified `KEY_STRING` as passphrase.

−−key `ID`:file:FILENAME
Read the passphrase from *FILENAME*.

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

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

−−no−applications
By default the output of virt-inspector includes the list of all the applications installed in the guest, if available.

Specify this option to disable this part of the resulting XML.

−−no−icon
By default the output of virt-inspector includes the icon of the guest, if available (see ‘‘icon’’).

Specify this option to disable this part of the resulting XML.

−v
−−verbose
Enable verbose messages for debugging.

−V
−−version
Display version number and exit.

−x Enable tracing of libguestfs API calls.

−−xpath query
Perform an XPath query on the XML on stdin, and print the result on stdout. In this mode virt-inspector simply runs an XPath query; all other inspection functions are disabled. See ‘‘XPATH QUERIES’’ below for some examples.

OLD-STYLE COMMAND LINE ARGUMENTS

Previous versions of virt-inspector allowed you to write either:

```
virt−inspector disk.img [disk.img ...]
```

or

```
virt−inspector guestname
```

whereas in this version you should use *−a* or *−d* respectively to avoid the confusing case where a disk image might have the same name as a guest.

For compatibility the old style is still supported.

XML FORMAT

The virt-inspector XML is described precisely in a RELAX NG schema file *virt−inspector.rng* which is supplied with libguestfs. This section is just an overview.

The top-level element is <operatingsystems>, and it contains one or more <operatingsystem> elements. You would only see more than one <operatingsystem> element if the virtual machine is multi-boot, which is vanishingly rare in real world VMs.

<operatingsystem>

In the <operatingsystem> tag are various optional fields that describe the operating system, its architecture, the descriptive ‘‘product name’’ string, the type of OS and so on, as in this example:

```
<operatingsystems>
  <operatingsystem>
    <root>/dev/sda2</root>
    <name>windows</name>
    <arch>i386</arch>
    <distro>windows</distro>
    <product_name>Windows 7 Enterprise</product_name>
    <product_variant>Client</product_variant>
    <major_version>6</major_version>
    <minor_version>1</minor_version>
    <windows_systemroot>/Windows</windows_systemroot>
```

In brief, <name> is the class of operating system (something like `linux` or `windows`), <distro> is the distribution (eg. `fedora` but many other distros are recognized) and <arch> is the guest architecture. The other fields are fairly self-explanatory, but because these fields are taken directly from the libguestfs inspection API you can find precise information from ‘‘INSPECTION’’ in **guestfs** (3).

The <root> element is the root filesystem device, but from the point of view of libguestfs (block devices may have completely different names inside the VM itself).

<mountpoints>

Un*x−like guests typically have multiple filesystems which are mounted at various mountpoints, and these are described in the <mountpoints> element which looks like this:

```
<operatingsystems>
  <operatingsystem>
    ...
    <mountpoints>
      <mountpoint dev="/dev/vg_f13x64/lv_root">/</mountpoint>
      <mountpoint dev="/dev/sda1">/boot</mountpoint>
    </mountpoints>
```

As with <root>, devices are from the point of view of libguestfs, and may have completely different names inside the guest. Only mountable filesystems appear in this list, not things like swap devices.

<filesystems>

<filesystems> is like <mountpoints> but covers *all* filesystems belonging to the guest, including swap and empty partitions. (In the rare case of a multi-boot guest, it covers filesystems belonging to this OS or shared with this OS and other OSes).

You might see something like this:

```
<operatingsystems>
  <operatingsystem>
    ...
    <filesystems>
      <filesystem dev="/dev/vg_f13x64/lv_root">
        <type>ext4</type>
        <label>Fedora-13-x86_64</label>
        <uuid>e6a4db1e-15c2-477b-ac2a-699181c396aa</uuid>
      </filesystem>
```

The optional elements within <filesystem> are the filesystem type, the label, and the UUID.

<applications>

The related elements <package_format>, <package_management> and <applications> describe applications installed in the virtual machine.

<package_format>, if present, describes the packaging system used. Typical values would be `rpm` and `deb`.

<package_management>, if present, describes the package manager. Typical values include `yum`, `up2date` and `apt`

<applications> lists the packages or applications installed.

```
<operatingsystems>
  <operatingsystem>
    ...
    <applications>
      <application>
        <name>coreutils</name>
        <version>8.5</version>
        <release>1</release>
      </application>
```

The version and release fields may not be available for some types guests. Other fields are possible, see ‘‘guestfs_inspect_list_applications’’ in **guestfs** (3).

<drive_mappings>

For operating systems like Windows which use drive letters, virt-inspector is able to find out how drive letters map to filesystems.

```
<operatingsystems>
  <operatingsystem>
    ...
    <drive_mappings>
      <drive_mapping name="C">/dev/sda2</drive_mapping>
      <drive_mapping name="E">/dev/sdb1</drive_mapping>
    </drive_mappings>
```

In the example above, drive C maps to the filesystem on the second partition on the first disk, and drive E maps to the filesystem on the first partition on the second disk.

Note that this only covers permanent local filesystem mappings, not things like network shares.

Furthermore NTFS volume mount points may not be listed here.

<icon>

Virt-inspector is sometimes able to extract an icon or logo for the guest. The icon is returned as base64−encoded PNG data. Note that the icon can be very large and high quality.

```
<operatingsystems>
  <operatingsystem>
    ...
    <icon>
      iVBORw0KGgoAAAANSUhEUgAAAGAAAABg[.......]
      [... many lines of base64 data ...]
    </icon>
```

To display the icon, you have to extract it and convert the base64 data back to a binary file. Use an XPath query or simply an editor to extract the data, then use the coreutils **base64** (1) program to do the conversion back to a PNG file:

```
base64 −i −d < icon.data > icon.png
```

XPATH QUERIES

Virt-inspector includes built in support for running XPath queries. The reason for including XPath support directly in virt-inspector is simply that there are no good and widely available command line programs that can do XPath queries. The only good one is **xmlstarlet** (1) and that is not available on Red Hat Enterprise Linux.

To perform an XPath query, use the *−−xpath* option. Note that in this mode, virt-inspector simply reads XML from stdin and outputs the query result on stdout. All other inspection features are disabled in this mode.

For example:

```
$ virt−inspector −d Guest | virt−inspector −−xpath '//filesystems'
<filesystems>
     <filesystem dev="/dev/vg_f13x64/lv_root">
       <type>ext4</type>
[...]

$ virt−inspector −d Guest | \
    virt−inspector −−xpath "string(//filesystem[@dev='/dev/sda1']/type)"
ext4

$ virt−inspector −d Guest | \
    virt−inspector −−xpath 'string(//icon)' | base64 −i −d | display −
[displays the guest icon, if there is one]
```

GETTING INSPECTION DATA FROM THE LIBGUESTFS API

In early versions of libguestfs, virt-inspector was a large Perl script that contained many heuristics for inspecting guests. This had several problems: in order to do inspection from other tools (like guestfish) we had to call out to this Perl script; and it privileged Perl over other languages that libguestfs supports.

By libguestfs 1.8 we had rewritten the Perl code in C, and incorporated it all into the core libguestfs API (**guestfs** (3)). Now virt-inspector is simply a thin C program over the core C API. All of the inspection information is available from all programming languages that libguestfs supports, and from guestfish.

For a description of the C inspection API, read ‘‘INSPECTION’’ in **guestfs** (3).

For example code using the C inspection API, look for *inspect−vm.c* which ships with libguestfs.

inspect−vm.c has also been translated into other languages. For example, *inspect_vm.pl* is the Perl translation, and there are other translations for OCaml, Python, etc. See ‘‘USING LIBGUESTFS WITH OTHER PROGRAMMING LANGUAGES’’ in **guestfs** (3) for a list of man pages which contain this example

code.

GETTING INSPECTION DATA FROM GUESTFISH

If you use the guestfish *−i* option, then the main C inspection API ‘‘guestfs_inspect_os’’ in **guestfs**(3) is called. This is equivalent to the guestfish command `inspect−os`. You can also call this guestfish command by hand.

`inspect−os` performs inspection on the current disk image, returning the list of operating systems found. Each OS is represented by its root filesystem device. In the majority of cases, this command prints nothing (no OSes found), or a single root device, but beware that it can print multiple lines if there are multiple OSes or if there is an install CD attached to the guest.

```
$ guestfish −−ro −a F15x32.img
><fs> run
><fs> inspect−os
/dev/vg_f15x32/lv_root
```

Using the root device, you can fetch further information about the guest:

```
><fs> inspect−get−type /dev/vg_f15x32/lv_root
linux
><fs> inspect−get−distro /dev/vg_f15x32/lv_root
fedora
><fs> inspect−get−major−version /dev/vg_f15x32/lv_root
15
><fs> inspect−get−product−name /dev/vg_f15x32/lv_root
Fedora release 15 (Lovelock)
```

Limitations of guestfish make it hard to assign the root device to a variable (since guestfish doesn’t have variables), so if you want to do this reproducibly you are better off writing a script using one of the other languages that the libguestfs API supports.

To list applications, you have to first mount up the disks:

```
><fs> inspect−get−mountpoints /dev/vg_f15x32/lv_root
/: /dev/vg_f15x32/lv_root
/boot: /dev/vda1
><fs> mount−ro /dev/vg_f15x32/lv_root /
><fs> mount−ro /dev/vda1 /boot
```

and then call the inspect-list-applications API:

```
><fs> inspect−list−applications /dev/vg_f15x32/lv_root | head −28
[0] = {
  app_name: ConsoleKit
  app_display_name:
  app_epoch: 0
  app_version: 0.4.5
  app_release: 1.fc15
  app_install_path:
  app_trans_path:
  app_publisher:
  app_url:
  app_source_package:
  app_summary:
  app_description:
}
[1] = {
  app_name: ConsoleKit−libs
  app_display_name:
```

```
  app_epoch: 0
  app_version: 0.4.5
  app_release: 1.fc15
  app_install_path:
  app_trans_path:
  app_publisher:
  app_url:
  app_source_package:
  app_summary:
  app_description:
}
```

To display an icon for the guest, note that filesystems must also be mounted as above. You can then do:

```
><fs> inspect-get-icon /dev/vg_f15x32/lv_root | display -
```

OLD VERSIONS OF VIRT-INSPECTOR

As described above, early versions of libguestfs shipped with a different virt-inspector program written in Perl (the current version is written in C). The XML output of the Perl virt-inspector was different and it could also output in other formats like text.

The old virt-inspector is no longer supported or shipped with libguestfs.

To confuse matters further, in Red Hat Enterprise Linux 6 we ship two versions of virt-inspector with different names:

```
virt-inspector     Old Perl version.
virt-inspector2    New C version.
```

EXIT STATUS

This program returns 0 if successful, or non-zero if there was an error.

SEE ALSO

guestfs (3), **guestfish** (1), http://www.w3.org/TR/xpath/, **base64** (1), **xmlstarlet** (1), http://libguestfs.org/.

AUTHORS

- Richard W.M. Jones http://people.redhat.com/~rjones/
- Matthew Booth mbooth@redhat.com

COPYRIGHT

LICENSE

BUGS

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

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

When reporting a bug, please supply:

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