NAME

p2v−hacking − extending and contributing to virt−p2v

DESCRIPTION

This manual page is for hackers who want to extend virt−p2v itself.

Virt−p2v is a front end on virt−v2v. ie. All it does is act as a GUI front end, and it calls out to virt−v2v to perform the actual conversion. Therefore most of the C code is Gtk (GUI) code, or supporting code for talking to the remote conversion server. There is no special support for physical machines in virt−v2v. They are converted in the same way as foreign VMs.

THE SOURCE CODE

Virt−p2v source is located in the github repository https://github.com/libguestfs/virt−p2v

Virt−p2v uses an autotools-based build system, with the main files being *configure.ac* and *Makefile.am*. See "THE BUILD SYSTEM".

To build from source, first read the **p2v−building** (1).

SOURCE CODE SUBDIRECTORIES

The majority of the source code is directly in the top level directory of the sources. There are also some subdirectories that contain some specific sub-parts of virt−p2v.

bash
Bash tab-completion scripts.

build-aux
Various build scripts used by autotools.

miniexpect
A copy of the miniexpect library from http://git.annexia.org/?p=miniexpect.git;a=summary.

contrib
Outside contributions, experimental parts.

docs
Miscellaneous manual pages.

gnulib
Gnulib is used as a portability library. A copy of gnulib is included under here.

libguestfs
Some sources (mostly with utilities) copied from libguestfs. Changes to the sources there ought to be forwarded to libguestfs as well.

m4 M4 macros used by autoconf. See "THE BUILD SYSTEM".

website
The virt−p2v files of the http://libguestfs.org website.

THE BUILD SYSTEM

Virt−p2v uses the GNU autotools build system (autoconf, automake).

The *./configure* script is generated from *configure.ac* and *m4/p2v−*.m4*. Most of the configure script is split over many m4 macro files by topic, for example *m4/p2v−libraries.m4* deals with the libraries required by virt−p2v.

subdir−rules.mk is included in every *Makefile.am* (top level and subdirectories).

UNDERSTANDING THE CODE

See also: "HOW VIRT−P2V WORKS" in **virt−p2v** (1)

There are two paths through the code, GUI or non-GUI (parsing the kernel command line):

```
 main.c  gui.c  conversion.c
               |                   |
               |                   |
              kernel.c
```

but both paths call back to the *conversion.c* function `start_conversion` to run the remote virt−v2v.

The main task of *gui.c*/*kernel.c* is to populate the virt−v2v configuration (*config.c*).

During conversion, we need to establish ssh connections, and that is done using two libraries:

```
 conversion.c  ssh.c  miniexpect.c
```

where *ssh.c* is responsible for managing ssh connections overall, and *miniexpect.c* implements ''expect-like'' functionality for talking interactively to the remote virt−v2v conversion server.

(Note that miniexpect is a separate library with its own upstream, so if you patch miniexpect.c, then please make sure the changes get reflected in miniexpect's upstream too: *http://git.annexia.org/?p=miniexpect.git;a=summary*)

RUNNING VIRT−P2V

You can run the *virt−p2v* binary directly, but it will try to convert your machine's real */dev/sda* which is unlikely to work well. However virt−p2v also has a test mode in which you can supply a test disk:

```
 make run−virt−p2v−directly
```

This is a wrapper around the **virt−p2v**(1) *−−test−disk* option. You can control the ''physical machine'' disk by setting `PHYSICAL_MACHINE` to point to a disk image.

A more realistic test is to run virt−p2v inside a VM on the local machine. To do that, do:

```
 make run−virt−p2v−in−a−vm
```

This also runs qemu with the ''physical machine'' disk (which you can set by setting `PHYSICAL_MACHINE`), a virtual CD, and a variety of network cards for testing. You can change the qemu binary and add extra qemu options by setting `QEMU` and/or `QEMU_OPTIONS` on the make commandline.

A third way to run virt−p2v simulates fairly accurately the program being downloaded over PXE and then doing an automatic conversion of the source physical machine (the non-GUI path — see next section below):

```
 make run−virt−p2v−non−gui−conversion
```

EXTENDING VIRT−P2V

FORMATTING CODE

Our C source code generally adheres to some basic code-formatting conventions. The existing code base is not totally consistent on this front, but we do prefer that contributed code be formatted similarly. In short, use spaces-not-TABs for indentation, use 2 spaces for each indentation level, and other than that, follow the K&R style.

If you use Emacs, add the following to one of your start-up files (e.g., ˜/.emacs), to help ensure that you get indentation right:

```
 ;;; In virt-p2v, indent with spaces everywhere (not TABs).
 ;;; Exceptions: Makefile and ChangeLog modes.
 (add−hook 'find−file−hook
     '(lambda () (if (and buffer−file−name
                          (string−match "/virt−p2v\\>"
                              (buffer−file−name))
                          (not (string−equal mode−name "Change Log"))
                          (not (string−equal mode−name "Makefile")))
                     (setq indent−tabs−mode nil))))

 ;;; When editing C sources in virt-p2v, use this style.
 (defun virt−p2v−c−mode ()
   "C mode with adjusted defaults for use with virt-p2v."
   (interactive)
   (c-set-style "K&R")
   (setq c-indent-level 2)
   (setq c-basic-offset 2))
 (add-hook 'c-mode-hook
           '(lambda () (if (string-match "/virt-p2v\\>"
                                        (buffer-file-name))
                           (virt-p2v-c-mode))))
```

TESTING YOUR CHANGES

Turn warnings into errors when developing to make warnings hard to ignore:

```
 ./configure --enable-werror
```

Useful targets are:

`make check`
: Runs the regular test suite.

  This is implemented using the regular automake `TESTS` target. See the automake documentation for details.

`make check-valgrind`
: Runs a subset of the test suite under valgrind.

`make check-slow`
: Runs some slow/long-running tests which are not run by default.

  To mark a test as slow/long-running:

  - Add it to the list of `TESTS` in the *Makefile.am*, just like a normal test.
  - Modify the test so it checks if the `SLOW=1` environment variable is set, and if *not* set it skips (ie. returns with exit code 77). If using `$TEST_FUNCTIONS`, you can call the function `slow_test` for this.
  - Add a variable `SLOW_TESTS` to the *Makefile.am* listing the slow tests.
  - Add a rule to the *Makefile.am*:

    ```
     check-slow:
       $(MAKE) check TESTS="$(SLOW_TESTS)" SLOW=1
    ```

VALGRIND

When you do `make check-valgrind`, it searches for any *Makefile.am* in the tree that has a `check-valgrind:` target and runs it.

Writing the *Makefile.am* and tests correctly to use valgrind and working with automake parallel tests is subtle.

If your tests are run via a shell script wrapper, then in the wrapper use:

```
 $VG virt-foo
```

and in the *Makefile.am* use:

```
 check-valgrind:
     make VG="@VG@" check
```

However, if your binaries run directly from the `TESTS` rule, you have to modify the *Makefile.am* like this:

```
 LOG_COMPILER = $(VG)

 check-valgrind:
     make VG="@VG@" check
```

In either case, check that the right program is being tested by examining the *valgrind** log files carefully.

SUBMITTING PATCHES

Submit patches to the mailing list: http://www.redhat.com/mailman/listinfo/libguestfs and CC to rjones@redhat.com.

You do not need to subscribe to the mailing list if you don't want to. There may be a short delay while your message is moderated.

SEE ALSO

p2v−building (1), **p2v−release−notes** (1), http://libguestfs.org/.

AUTHORS

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

COPYRIGHT

LICENSE

BUGS

To get a list of bugs against libguestfs (which include virt−p2v), 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 virt−p2v.
- Where you got virt−p2v (eg. which Linux distro, compiled from source, etc)
- Describe the bug accurately and give a way to reproduce it.