Perl script cleans up virtual machines

Freshen Up

A VirtualBox installation creates snapshots of virtual machines in next to no time at the command line; it also helps protect your privacy while browsing and sends Perl modules through a smoke test. *By Mike Schilli*

If you are continually trying out new Linux distributions, you will probably be able to use a virtualizer like VMware, KVM, Xen, or VirtualBox blindfolded. But programmers who need to develop code for different Linux versions or, heaven forbid, Windows versions, also appreciate locally available virtual machines for quick tests.

The VirtualBox [2] virtualization package is easily controlled through its attractive GUI (Figure 1) and is easily downloaded as an Ubuntu package under the GPLv2 license (Figure 2).

Users can quickly create a handful of guest systems using installation CD/DVDs or ISO files and boot the guests in separate windows at the press of a button. Mouse control on the guest system needs some getting used to; once the guest system has grabbed the mouse focus, it doesn't let go. The mouse pointer won't move outside of the guest system window borders, except for some guest systems that allow a shared focus. To leave the virtual system focus, you need to press a predefined key (the default is the right Alt key); this takes the mouse pointer back to your desktop.

Freezing and Thawing

Efficient snapshot technology lets you freeze a virtual machine's state and re-

Figure 1: VirtualBox with some Ubuntu and Windows versions as guest systems.

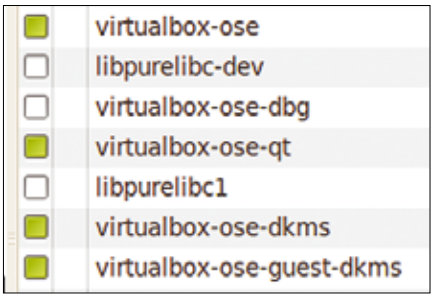

Figure 2: Synaptic listing of Ubuntu packages that install VirtualBox.

MIKE SCHILLI

Mike Schilli works as a software engineer with Yahoo! in Sunnyvale, California. He can be contacted at *mschilli@perl-meister.com*. Mike's homepage can be found at *http://perlmeister.com*.

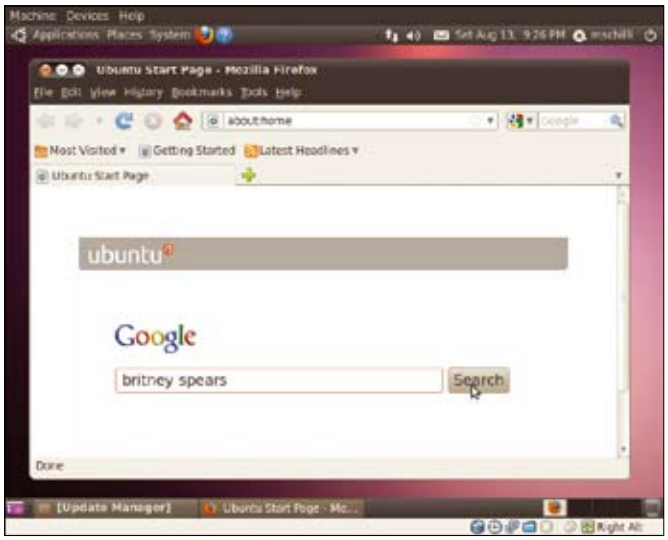

Figure 3: The browser will have forgotten this embarrassing search term the next time you fire up the virtual machine.

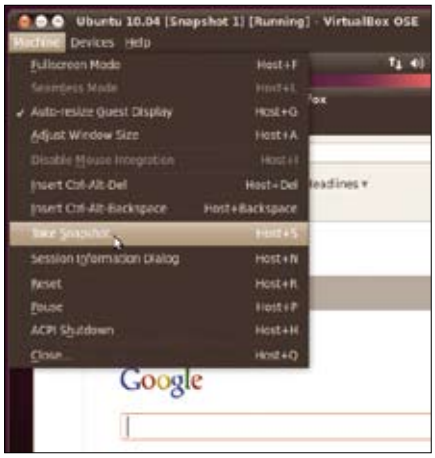

Figure 4: The user creating a snapshot of the virtual machine state.

store it in a couple of seconds. This means that web surfers can use a freshly installed Ubuntu with an open browser for embarrassing web searches (Figure 3).

The torrid details of what Google knows about its users are always revealed when you enter a search key and the autocomplete function does the rest. After all, nobody wants to be reminded that they searched for "athlete's foot" just a couple of days ago, not to mention the influence that the search term has on longer-term targeting by Google's personalized advertising machine.

Paranoid penguin friends will not want to keep the cookies they received for longer than necessary or to leave any tracks in the browser history. Because there's no way of knowing exactly what the browser does behind your back in terms of caching and other data storage, it only makes sense to fire up a virtual machine quickly for sensitive searches and then to restore the virtual machine's original status when you're done.

This approach isn't entirely watertight; if you're combating a mastermind, they might find data snippets on the virtual hard disk, and the IP address of your router will appear in Google's logs, but it does raise the bar considerably. If you need more privacy, you might like to use Tor [3] and also run shred against the virtual machine disk, which is a single file.

Shortly after installing Linux from the DVD, users who are interested in protecting their privacy will launch the Firefox browser in the brand-new guest system and create a snapshot of the virtual machine (Figure 4).

After completing his secret mission, the user then powers off the virtual machine using the *Stop* command and then switches back to the original state by selecting the *Restore* function in the VirtualBox snapshot menu. The next time you launch the virtual machine you will have an open browser that doesn't remember a thing – just as if you'd jumped back in time.

Automated Command

Instead of clicking around in the menus to launch the VirtualBox GUI every time you want to research something, you might prefer to use a Perl script that automates the process of selecting and booting the virtual machine. When you're done, you press the Enter key in the script to shut down the virtual machine and restore its original state. Luckily, VirtualBox offers the `VBoxManage` tool, which gives you complete control at the command line. Listing 1 shows the Perl script that selects the virtual machine named "Ubuntu 10.04" and its "Browse" snapshot, which I prepared previously (see the list of snapshots in Figure 1).

The script uses the `tap` command from the CPAN Sysadm::Install module to issue shell commands. Of course, it might be easier to implement the program as a shell script, but experienced Perl programmers know it's only a matter of time until the functional scope of a shell script has grown to the extent that you will want to reimplement it in Perl for fear of not being able to maintain the monster.

The script uses the subcommands `startvm` and `ctrlvm poweroff` provided by `VBoxManage` to start and shut down the virtual machine. It restores a snapshot both at the start and at the end of the script to make quite sure that the virtual machine boots into the snapshotted "Browse" state, even if somebody has messed around in the Snapshot menu of the VirtualBox GUI in the meantime.

Limited Networking

VirtualBox virtual machines can also be managed in headless mode without any screen output. An invisible SSHD dae-

LISTING 1: browse

```
01 #!/usr/local/bin/perl -w
02 ##############################
03 # browse - VM for browsing
04 # Mike Schilli, 2011
05 # (m@perlmeister.com)
06 ##############################
07 use strict;
08 use Sysadm::Install qw(:all);
09 use Log::Log4perl qw(:easy);
10
11 Log::Log4perl->easy_init(
12  $DEBUG);
13
14 my $vbm = "VBoxManage";
15 my $vm  = "Ubuntu 10.04";
16
17 tap $vbm, "snapshot", $vm,
18   "restore", "Browse";
19 tap $vbm, "startvm", $vm;
20
21 print
22   "Press Enter for shutdown";
23 <STDIN>;
24
25 tap $vbm, "controlvm", $vm,
26   "poweroff";
27 tap $vbm, "snapshot", $vm,
28   "restore", "Browse";
```

mon running on the guest system lets you execute commands from the host system or exchange files between guest and host.

That said, VirtualBox launches guest systems in NAT (Network Address Translation) mode by default, assigning them an address on the virtual 10.x.x.x network, and using address translation to communicate with the host's local network in a similar approach that most routers in private homes take to let the local devices with their 192.168.x.x addresses communicate with the Internet. This works fine from the virtual machine to the local network; however, the host and the devices on the local network can't open a connection to the virtual machine. The Network Adapters dialog hidden behind the icon with the two terminals bottom right on an active virtual machine (Figures 5 and 6) allows you to change this behavior.

Bridge Opens Firewall

If you change the setting from *NAT* to *Bridged Mode*, the virtual machine retrieves an IP address from the DHCP server on the local network (e.g., 192.168.0.135), becoming a peer communication partner. If you launch an SSHD daemon with

```
sudo apt-get ➥
    install ➥
    openssh-server
```

on the virtual machine, clients on the host or on the local network that log in to the virtual machine (`ssh 192.168.0.135`) with their user IDs are granted access. If the user also copies their SSH public key (e.g., `~/.ssh/id_rsa.pub`) to the `~/.ssh/authorized_keys` file on the virtual machine, they don't even need a password – and this is important for automated scripts, assuming the private key was created without a passphrase.

Figure 5: Right-clicking the icon with the two terminals leads to a dialog box for setting up the network adapter.

Figure 6: The Bridged Adapter in the network dialog supports bidirectional communications between the virtual machine and the rest of the world.

Figure 7: Installing Guest Additions gives you useful utilities for VirtualBox VMs.

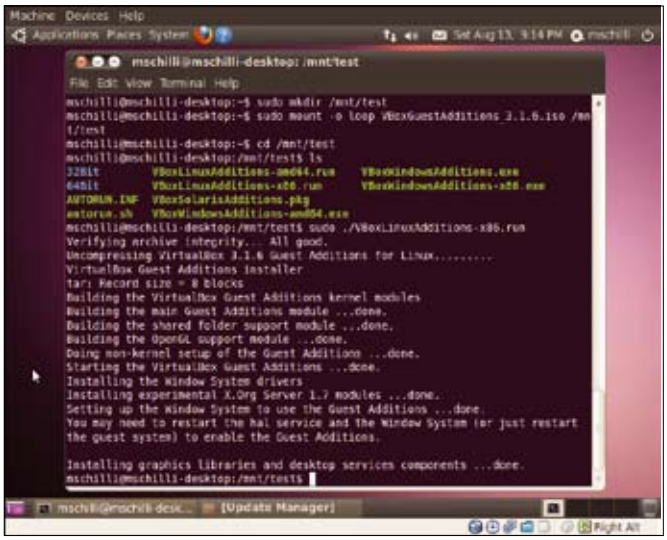

Figure 8: Installing the Guest Additions on 10.04 takes a fair bit of manual TLC.

However, it's not always easy to find out which IP address a specific virtual machine has picked up when it launches. VirtualBox offers a method via guest properties; an extension that you first need to install on the active virtual machine after selecting *Devices* | *Install Guest Additions* (Figure 7).

After doing so, the virtual machine downloads an ISO file off the Internet, mounts the file like a CD drive on the virtual machine, and executes a shell script stored on the CD that triggers an orgy of kernel module builds.

Manual Attention Needed

Unfortunately, this approach didn't work in the VirtualBox version that I was using, 3.1.6; I had no alternative but to download `VBoxGuestAdditions_3.1.6.iso` manually to the virtual machine, mount the file using a `mount -o loop`, install the `dkms` Ubuntu package, and then execute the `VBoxLinuxAdditions-x86.run` shell script from the command line (Figure 8). Note that this will not work with Ubuntu 11 as the guest system if the host system is still running Ubuntu 10.04, which seems to require platform parity.

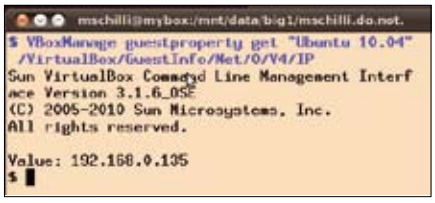

Figure 9: The guestproperty command shows you which IP address the specific virtual machine has picked up from the DHCP server.

After restarting the virtual machine, the host can issue the `VBoxManage guestproperty enumerate "Ubuntu 10.04"` command and, hidden in a mass of other information, receive the line `"Name: /VirtualBox/GuestInfo/Net/0/V4/IP, value: 192.168.0.135"`, which gives you the IP used by the virtual machine. If you run `get` for the guestinfo path shown above instead of `enumerate`, you are sent the IP address of the active virtual machine directly (Figure 9).

Smoking Chimneys

Another use case for virtual machines is testing CPAN modules in pristine environments. Assuming CPAN modules do a good job of specifying their dependencies on other modules, a CPAN shell will handle the installation without breaking a sweat. Unfortunately, some authors forget to specify modules they have on their development machines that a vanilla Perl installation will not include. This causes much frustration among users – except for cases in which one of today's ubiquitous and automatically activated CPAN smoke tests identify a problem and notify the author via email.

A Perl installation stored as a virtual machine snapshot that the module programmer can boot with the `cpan-smoke` script (Listing 2) is a big help here. You just pass in your newly created CPAN distribution tarball and let a CPAN shell test whether the new version is installable on the vanilla system with a connection to CPAN. If the module survives the accompanying test suite without error, it passes the smoke test and you have a good chance that the module will install on similar systems.

Branches on the Snapshot Tree

For this to happen, developers need to configure a CPAN shell on a virtual machine with retrospectively installed Guest Additions, create a snapshot, and call the snapshot "CPAN Smoke." Figure 10 shows two branches on the snapshot tree of the Ubuntu 10.04 virtual machine.

The originally created snapshot, Browse, inherits from a fresh installation and is independent of any snapshots you might create on the basis of the second branch, CPAN Smoke.

To make sure the installation also works with the mini CPAN shell `cpanm` in the local directory, the CPAN Smoke snapshot also installs the CPAN local::lib module. The best way of doing this on an Ubuntu system is to issue the command

```
$ sudo apt-get install liblocal-lib-perl
```

which relies on the package manager to install local::lib on the virtual machine in the main perl module tree only accessible by root. The command

```
$ eval $(/usr/bin/perl -Mlocal::lib)
```

issued in the shell sets the shell variables required to install additional CPAN modules in the user's home directory, accessible by the user's regular ID. You can install the mini-shell locally via the "big" CPAN shell:

```
$ cpan App::cpanminus
```

Figure 10: A new CPAN Smoke snapshot, derived from a vanilla Ubuntu install with the Guest Additions, plus a configured and ready-to-run CPAN shell.

After doing so, the `cpanm` command for installing other modules will be available in your `$PATH`:

```
$ cpanm --version
cpanm (App::cpanminus) version 1.4008
```

Now you create the snapshot and store it as "CPAN Smoke." If you then call `cpan-smoke` on the host and pass in a CPAN tarball created with `make tardist` as an argument, the script will restore

Layout: Listing 2, top of column 1 looks a bit crowded. -rls

LISTING 2: cpan-smoke

```
01 #!/usr/local/bin/perl -w
02 ############################
03 # cpan-smoke - VM for CPAN
04 #   module smoke testing
05 # Mike Schilli, 2011
06 # (m@perlmeister.com)
07 ############################
08 use strict;
09 use Sysadm::Install qw(:all);
10 use Proc::Simple;
11 use Net::Ping;
12 use Log::Log4perl qw(:easy);
13
14 my ($tarball) = @ARGV;
15
16 die "usage: $0 tarball"
17   if !defined $tarball;
18
19 Log::Log4perl->easy_init(
20  $ERROR);
21
22 my $vbm = "VBoxManage";
23 my $vbh = "VBoxHeadless";
24 my $vm  = "Ubuntu 10.04";
25 my $ippath = "/VirtualBox" .
26     "/GuestInfo/Net/0/V4/IP";
27 my $snap = "CPAN Smoke";
28
29   # in case it's up in
30   # foreground mode
31 tap $vbm, "controlvm", $vm,
32   "poweroff";
33 tap $vbm, "snapshot",
34   $vm, "restore", $snap;
35
36 my $proc =
37   Proc::Simple->new();
38 $proc->start(
39  "$vbh --startvm '$vm'");
40
41 END {
42  $proc->kill();
43 }
44
45 INFO
46  "Waiting for VM to come up";
47
48 while (!$proc->poll()) {
49  DEBUG
50    "Waiting for VM process";
51  sleep 1;
52 }
53
54 my $ip;
55
56 while (!defined($ip = ip()))
57 {
58  DEBUG "Waiting for IP";
59  sleep 1;
60 }
61
62 my $ping = Net::Ping->new();
63 while (!$ping->ping($ip)) {
64  DEBUG "Waiting for Ping";
65  sleep 1;
66 }
67
68 INFO "VM is up: $ip";
69
70 tap "scp", $tarball,
71   "$ip:/tmp/$tarball";
72 sysrun "ssh", $ip,
73 qq{eval \$(/usr/bin/perl
      -Mlocal::lib);
74    cpanm /tmp/$tarball};
75
76 tap $vbm, "controlvm", $vm,
77   "poweroff";
78 tap $vbm, "snapshot",
79   $vm, "restore", $snap;
80
81 ############################
82 sub ip {
83 ############################
84  my ($stdout) = tap $vbm,
85    "guestproperty",
86    "get", $vm, $ippath;
87
88  if (
89   $stdout =~ /Value: (.*)/)
90  {
91   return $1;
92  }
93
94  return undef;
95 }
```

the snapshot, start the virtual machine, and wait until its network configuration is complete and responds to `ping`.

The `ip()` function (Listing 2, lines 82-95) looks for the IP address of the virtual machine you are using by sending a `guestproperty` request to VirtualBox. A regular expression grabs the relevant IP address from the verbose output.

The `VBoxHeadless --startvm` command of line 39 launches the virtual machine without any GUI output and waits in the foreground until the user kills the program by pressing Ctrl+C. The script uses the CPAN Proc::Simple module to send the `starvm` command into the background, and remembers its PID so that the `END` snippet triggered at the end of the script sends its `kill()` command to the right program in line 42, thus shutting down the headless virtual machine.

Smoke Signals?

Line 70 copies the tarball into the virtual machine's `/tmp` directory, and line 74 launches the `cpanm [tarball]` command, which picks up the tarball, unpacks it, and issues `make` commands to test and install. If `cpanm` discovers CPAN modules that the module specifies as dependencies, it downloads them automatically from CPAN and proceeds to install them.

To make sure the shell in the virtual machine finds the `cpanm` command, searches for installed CPAN modules in the local `~/perl5` path, and installs new modules in this path, the `ssh` command in line 72 issues the `eval` command shown earlier to set the required shell variables before running the `cpanm` command itself.

The output from the `cpanm` installation script is directed to the standard output of the calling `cpan-smoke` script, thanks to the Sysadm::Install module's `sysrun` function, pointing diligent developers to first-hand information on how their latest creation behaves in an pristine environment. If you do see a smoke signal, you can assume something is wrong; most likely, the dependencies on other CPAN modules need careful review.

Even in today's age of the cloud, where virtual machines such as Amazon's EC2 are available on the web for those on a budget, a local virtualization solution like VirtualBox is still an attractive option. A detailed instruction manual for newcomers – compared with the typically less structured offerings on the Internet – is available [4], although it describes the more or less obsolete VirtualBox version 3.1. ■■■

INFO

[1] Listings for this article: *http://www.linuxpromagazine.com/Resources/Article-Code*

[2] VirtualBox Project Page: *http://www.virtualbox.org*

[3] Tor: *http://www.torproject.org/*

[4] Romero, Alfonso V. *VirtualBox 3.1: Beginner's Guide*. Packt Publishing, 2010 (Kindle edition)