Preface

Qemu is a popular and powerful open-source emulator often used for running KVM Virtual Machines (VMs). In fact qemu supports emulating so many things that it can be quite challenging, unless you do it very often, to manually start a VM from a text console. Who would want to write the below command for starting a VM ?

```
qemu-system-arm -name armedslack -M versatilepb -m 256 -k en-us -vnc
:5,password -usb -kernel /VM/armedslack/zImage-versatile -initrd
/VM/armedslack/initrd-versatile -append 'root=/dev/sda1 rootfs=ext2' -
monitor telnet:127.0.0.1:1035,server,nowait  -drive
file=/VM/armedslack/qemu_hdu.raw,index=0,media=disk -drive
file=,index=1,media=cdrom  -net nic,macaddr=52:54:57:c0:2c:bb -net
tap,ifname=tap5
```

Not everyone might want console redirect on vnc and monitor redirect via telnet but non the less that's still a relatively small subset of the options supported by qemu-system-arm and only has one disk one cdrom and one Network Interface Controller (NIC) so things can be much worse then this.

Is is common, for people running qemu VMs, to use some sort of software for creating, running and maintain the VMs. If you want an easy way out you might consider virtmanager or something like that. Personally I chose to manage my VMs from text console because for me it's an added value to be able to do such jobs even without GUI, so possibly like many others I wrote my onw scripts for managing my qemu VMs.

Over the years I've radically changed the helper script form having text configuration files for each VM to a centarl VM configuration database. I'd like to share my experience in doing so without presumptuously declaring that I do this any better then anyone else, letting you decide what's good or bad for your needs. It's likely that someone else has done this and a lot better then me but nevertheless I'd still like to hare with you the route I took.

Problems

Here are the problems that governed my choices:

- ability to run both x86 and ARM virtual machines
- ability to run several VMs simultaneously
- have the VMs appear as a real server in the LAN
- flexibility on the number of disks assigned to a VM
- flexibility on the number of NICs assigned to a VM
- avoid conflicts on VM console vnc port
- avoid conflicts on VM monitor port

Wanting the VMs to appear like real servers on the LAN, along with my other requirements, made me opt for bridged tap NICs, which in turn created another potential conflict on the tap device ans MAC address.

Using the initial, per VM text based configuration, started to make it difficult to deal with such issues automatically (making the code unnecessarily long and complex) while manually creating a configuration file for a new VM required looking for information across all previously configured VMs to avoid potential conflicts.

Proposed Solution

It quickly became apparent to me that the VM configuration would need to be generated rather then manually created and that a central configuration repository would much aid the process. Again a text based central configuration file would make, either the code or the config file, inherently complicated (having to deal with an arbitrary number of VMs each with arbitrary number of disks and NICs). Having some experience on database administration made it a little unappealing to use LDAP for central repository and even if I had no DB experience at all I doubt I'd actually want the overhead of running LDAP just for this. Running MariaDB or Postgres was equally unappealing too for my small requirements, so I chose to use sqlite3. In my case it would be extremely rare that, 2 or more simultaneous executions of the management script, make a mess on the DB but if you have several people managing creating or deleting VMs you might want to opt for MariaDB or Postgres.

Another thing that quickly became apparent was the almost repetitive code required to prompt for all the options so I decided to address that in 2 ways:

1. have as much of the promoting automatically generated with a clever workaround
2. use dialog to further simplify the UI for prompting

Basic Configuration

To get better flexibility for configuring where things are stored it's a good idea to have a basic configuration file that tells the management script where the important things are:

- Path to folder that will contain all the VMs
- Path to where the centralized VM configuration DB is
- Path to where the ISO images are stored

I also find it handy to have ither parameters in the basic configuration:

- MONITOR_START_PORT
- SHUTDOWN_TIMEOUT (timeout for clean shutdown od a VM)
- OS_RESERVED_MEM (how much memory is to be reserved for OS when prompting for how much RAM is to be assigned to a VM)

Here's an example of what the config file would look like:

```
VM_PATH='/VM'
VM_DB='/VM/vm.conf'
ISO_PATH='/ISO'
MONITOR_START_PORT='1030'
```

```
SHUTDOWN_TIMEOUT='300'
OS_RESERVED_MEM='512'
```

Getting the Management Script to Find the Basic Configuration

To make sure the management script can find this basic configuration no matter what I use a simple workaround: have the basic config in the same place as the management script and call it the same with ".conf" appended to it.

For example let's suppose the management script is **/usr/local/bin/qemumgmt** it's configuration file would be **/usr/local/bin/qemumgmt.conf** thus inside the management script I can load the basic configuration like this (or get the script to prompt for making one):

```
NAME=$(basename $0)
CWD=$(dirname $0)
[ -r ${CWD}/${NAME}.conf ] && source ${CWD}/${NAME}.conf || make_config
```

A smart Way to Produce Basic Configuration

Getting someone to use your script if it's difficult to configure is difficult so I opted for having the script prompt the user for it's mandatory configuration options and automatically create the basic configuration file if it's not there. I do this by using the **make_config** finction to which I refer th the above paragraph and a variable that holds the mandatory parameter list (there are also 2 function for manipulating input and generating a dialogs based on the input **form_dialog** and **accept_dialog**):

```
SCRIPT_PARAMS="VM_PATH VM_DB ISO_PATH MONITOR_START_PORT SHUTDOWN_TIMEOUT
OS_RESERVED_MEM"

form_dialog ()
{ TITLE=$1
  shift
  unset STRING
  MAX=0
  for A in $*
  do
    [ $((${#A} +3))  -gt $MAX ] && MAX=$((${#A} +3))
  done
  i=0
  for A in $*
  do
    STRING[$i]="${A}: $(($i + 1))  1  \"$(eval echo "\$$A")\" $(($i + 1))
$MAX 40 0"
    ((i++))
  done
  eval $(echo dialog --title \"$TITLE\" --separator \'\;\' --form \"\" 0 0 0
${STRING[*]} \2\>$DIALOG_OUTPUT )
}
```

```
accept_dialog ()
{ TITLE=$1
  shift
  unset STRING
  MAX=0
  for A in $*
  do
    [ $((${#A} +3))  -gt $MAX ] && MAX=$((${#A} +3))
  done
  i=0
  for A in $*
  do
    STRING[$i]="${A} = $(eval echo "\$$A") \n"
    ((i++))
  done
  eval $(echo dialog --title \"$TITLE\" --yesno  \" ${STRING[*]}\" 0 0 ) &&
return 1 || return 0
}

make_config ()
{ ACCEPT=0
  while [ $ACCEPT -eq 0 ]
  do
    form_dialog "Basic config for $NAME" $SCRIPT_PARAMS && IFS=';' read -r -
d '\;' $SCRIPT_PARAMS < $DIALOG_OUTPUT
    accept_dialog "Accept Configuration ?" $SCRIPT_PARAMS
    ACCEPT=$?
  done

  ( for PARAM in $SCRIPT_PARAMS
    do
      echo "${PARAM}='$(eval echo "\$$PARAM")'"
    done
  ) > $CWD/${NAME}.conf
  chmod 600 $CWD/${NAME}.conf
  sleep 1
}
```

Isn't that nice? there's no specific code for each basic configuration option ... everything is self generated from the **SCRIPT_PARAMS** variable. If you need to add another basic configuration option it's really fast ... and as long as you give them names that speak for themselves it will also be easy to give them values when prompted via dialog.

This is what it would be like to get propted (with defaults) for the missing basic configuration:

```
             +-----------------Basic config for qemu_dialog-----------------
---+
             | +-------------------------------------------------------------
-+ |
```

```
              | |VM_PATH:            /VM
| |
              | |VM_DB:              /VM/vm.conf
| |
              | |ISO_PATH:           /ISO
| |
              | |MONITOR_START_PORT: 1030
| |
              | |SHUTDOWN_TIMEOUT:   300
| |
              | |OS_RESERVED_MEM:    512
| |
              | +-----------------------------------------------------------
-+ |
              |
|
              +-------------------------------------------------------------
---+
              |                    <  OK  >          <Cancel>
|
              +-------------------------------------------------------------
---+
```

Configuration Database

Having opted for a DB central configuration would require that all the items with arbitrary quantities per VM be on a separate table that references a parent VM table. My issue was on having arbitrary number of disks and NICs per VM, all the other stuff could be held in columns of a master VM table. Let's have a look at the tables

Master VM Table

In this table you will need to have all the biunique features of each VM expressly leaving out all features that do not have a one to one (VM ↔ feature) relationship. As mentioned before I want to have the possibility that each VM has both several disks and NICs so these features will be in separate tables for me. So the features that are required and biunique to each VM in my needs are:

- Name
- System Emulator
- Machine Type
- CPU
- RAM
- Keyboard Layout
- Bios
- Boot Order
- Flag for KVM support
- Display (in case console is to be redirected)
- VNC Password

- Flag for USB
- Kernel Image
- Initrd Image
- Append Parameters for Kernel

Naturally to that we need to add a key for referencing the non biunique features of each VM. Here's the sql syntax for creating my VM master table (in sqlite3 dialect):

```
create table if not exists virtual_machines
( vm_id         integer primary key autoincrement,
  vm_name       text unique not null,
  vm_emulator   text not null,
  vm_machine    text not null,
  vm_cpu        text not null,
  vm_mem        integer not null,
  vm_kb         text not null,
  vm_bios       text,
  vm_boot_order text,
  vm_kvm        intger not null,
  vm_display    text,
  vm_vnc_pw     text,
  vm_usb        integer not null,
  vm_kernel     text,
  vm_initrd     text,
  vm_append     text
);
```

Disks Table

This table will need to hold a column that ties (references) which VM owns the disk along with the infomation required to identify the disk:

- Media Type (disk or cdrom)
- Media File (where the disk image file is located on the host machine)
- Status Flag (allow the VM to boot without the cdrom for example)

Here's the sql in sqlite3 dialect:

```
CREATE TABLE disks
( d_id          integer primary key autoincrement,
  d_vm_id       integer  references virtual_machines(vm_id) on update
cascade,
  d_media       text not null,
  d_file        text not null,
  d_status      integer not null
);
```

NICs Table

Like the disks table this also needs to have a column that references the VM that owns the NIC along with the other information required to identify the NIC itself: ID (used to have unique tap devices) MAC Address

Here's the sql in sqlite3 dialect:

```
CREATE TABLE nics
( n_id          integer primary key autoincrement,
  n_vm_id       integer  references virtual_machines(vm_id) on update
cascade,
  n_mac         text not null
);
```

Real Data

Here's what the data inside my configuration database looks like:

```
sqlite> select * from virtual_machines;
1|ORACLE|qemu-system-i386|pc|kvm32|1024|en-
us|file=/usr/share/qemu/bios.bin|c|1|vnc|ciccio|1|||
2|test|qemu-system-i386|pc|kvm32|1024|en-
us|file=/usr/share/qemu/bios.bin|c|1|vnc|ciccio|1|||
3|ORACLE2|qemu-system-i386|pc|kvm32|1024|en-
us|file=/usr/share/qemu/bios.bin|c|1|vnc|ciccio|1|||
4|freenas|qemu-system-x86_64|pc|host|3000|en-
us|file=/usr/share/qemu/bios.bin|c|1|vnc|pafutometu|1|||
5|armedslack|qemu-system-arm|versatilepb||256|en-
us|||0|vnc|cicciobello|1|/VM/armedslack/zImage-
versatile|/VM/armedslack/initrd-versatile|root=/dev/sda1 rootfs=ext2
sqlite> select * from disks;
1|1|disk|/VM/ORACLE/disk0.qcow2|1
2|1|cdrom|/ISO/OL6.6-x86.iso|1
3|2|disk|/VM/test/disk0.qcow2|1
4|2|cdrom|/ISO/OL6.6-x86.iso|1
5|1|disk|/VM/ORACLE/disk1.qcow2|1
6|3|disk|/VM/ORACLE2/disk0.qcow2|1
7|3|cdrom|/ISO/OL6.6-x86.iso|1
8|4|disk|/VM/freenas/disk0.qcow2|1
9|4|disk|/VM/freenas/disk1.qcow2|1
10|5|disk|/VM/armedslack/qemu_hdu.raw|1
11|5|cdrom||1
sqlite> select * from nics;
1|1|52:54:54:e7:30:88
2|2|52:54:54:ec:97:85
3|3|52:54:54:f6:cd:f9
4|4|52:54:57:be:b5:2e
5|5|52:54:57:c0:2c:bb
```

```
sqlite>
```

There is however a problem with using foreign keys with sqlite3, sometimes when you delete rows and the referenced rows in the PARENT table, the internal sqlite_sequence table can enter an inconsistent state inhibiting any further insertions on the child tables. This can be worked around by careful handling of both the tables and the internal sqlite_sequence table.

A Smart Way to Input Data For New VM

As mentioned above, getting bugged with all the details required for configuring a new VM can be a a killer unless you do it really often so here's how I go about it in a similar fashion to creating the basic configuration. The idea is to reduce the amount of code written specifically to each VM option allowing for a relatively slender script and ease to add new VM options if required.

Unfortunately here things are a little more complicated because different qemu-system-* produce slightly different output when prompted with help and because most of the VM options require a separate dialog asking for some specific action nonetheless a lot of the code required is self generated from the **PARAMS** variable. The only specific code is to work around the differences between the various qemu-system-* emulators.

```
PARAMS="EMULATOR MACHINE CPU MEM KEYBOARD DISK DISKSIZE CDROM BIOS
BOOT_ORDER KVM DISPLAY VNCPW USB KERNEL INITRD APPEND MAC"

create_vm ()
{ if [ $# -lt 1 ]
  then
    while [ "$VM_NAME" = "" ]
    do
      dialog --inputbox "Choose Virtual Machine Name
N.B. The following names are already in use
$(list_vm_names)
" 0 0  2>$DIALOG_OUTPUT
      VM_NAME=$(< $DIALOG_OUTPUT)
    done
  else
    VM_NAME=$1
  fi
  get_vm_id $VM_NAME
  VM_ID=$?
  while [ $VM_ID -ne 0 ]
  do
     dialog --inputbox "$VM_NAME is already in use, pick another one
N.B. The following names are already in use
$(list_vm_names)
" 0 0 2>$DIALOG_OUTPUT
    VM_NAME=$(< $DIALOG_OUTPUT)
    get_vm_id $VM_NAME
    VM_ID=$?
```

```
  done

  radiobox_dialog "Choose the type of system to be emulated" $(find
${PATH//:/ } -type f -executable -name "qemu-system-*" -printf "%f ")
  EMULATOR=$(< $DIALOG_OUTPUT)
  case $EMULATOR in
  qemu-system-i*86|qemu-system-x86_64)
    MACHINE=pc
    grep -qwE "vmx|svm" /proc/cpuinfo && KVM=1 || KVM=0
    [ $KVM -eq 1 ] && CPU=host || CPU=pentium3
    BIOS="file=/usr/share/qemu/bios.bin"
    BOOT_ORDER="c"
  ;;
    qemu-system-arm)
    KVM=0
    choose_machine $EMULATOR
    MACHINE=$(< $DIALOG_OUTPUT)
    choose_arm_cpu $MACHINE
    CPU=$(< $DIALOG_OUTPUT)
  ;;
  *) echo "unsupported yet" ; exit 1 ;;
  esac

  choose_mem_size
  MEM=$(< $DIALOG_OUTPUT)
  choose_keyboard
  KEYBOARD=$(< $DIALOG_OUTPUT)
  KEYBOARD=${KEYBOARD:-"en-us"}
  choose_iso
  CDROM=$(< $DIALOG_OUTPUT)
  choose_file $VM_PATH/$VM_NAME "Choose VM kernel"
  KERNEL=$(< $DIALOG_OUTPUT)
  if [ "$KERNEL" != "" ]
  then
    choose_file $VM_PATH/$VM_NAME "Choose VM initrd"
    INITRD=$(< $DIALOG_OUTPUT)
    choose_append
    APPEND=$(< $DIALOG_OUTPUT)
    unset BIOS
    unset BOOT_ORDER
  fi

  choose_file $VM_PATH/$VM_NAME "Choose VM disk (cancel for default)"
  DISK=$(< $DIALOG_OUTPUT)
  if [ "$DISK" = "" ]
  then
    DISK="$VM_PATH/$VM_NAME/disk0.qcow2"
    dialog --inputbox "Choose VM disk image size " 0 0 20G 2>$DIALOG_OUTPUT
    DISKSIZE=$(< $DIALOG_OUTPUT)
  fi
```

```
  dialog --inputbox "Type VNC password for this VM" 0 0  cicciobello
2>$DIALOG_OUTPUT
  VNCPW=$(< $DIALOG_OUTPUT)

  DISPLAY=vnc
  USB=1
  MAC="52:54:$(date +%s |awk '{printf("%x",($1 / 16777216)%256)}'):$(date
+%s |awk '{printf("%x",($1 / 65536)%256)}'):$(date +%s |awk
'{printf("%x",($1 / 256)%256)}'):$(date +%s |awk '{printf("%x",$1 % 256)}')"

  ACCEPT=0
  while [ $ACCEPT -eq 0 ]
  do
    form_dialog "Configuration for $VM_NAME VM" $PARAMS && IFS=';' read -r -
d '\;' $PARAMS < $DIALOG_OUTPUT
    accept_dialog "Accept $VM_NAME configuration ?" $PARAMS
    ACCEPT=$?
  done

  NEW_VM_ID=$(( $(count_configured_vms) + 1 ))
  echo -e "PRAGMA foreign_keys = ON;
insert into virtual_machines values
(null,'$VM_NAME','$EMULATOR','$MACHINE','$CPU',$MEM,'$KEYBOARD','$BIOS','$BO
OT_ORDER',$KVM,'$DISPLAY','$VNCPW',$USB,'$KERNEL','$INITRD','$APPEND');
insert into disks values (null,$NEW_VM_ID,'disk','$DISK',1);
insert into disks values (null,$NEW_VM_ID,'cdrom','$CDROM',1);
insert into nics values (null,$NEW_VM_ID,'$MAC');" |sqlite3 $VM_DB
[ ! -d $(dirname $DISK) ] && mkdir -p $(dirname $DISK)
[ ! -f $DISK ] && qemu-img create -f qcow2 -o preallocation=metadata $DISK
$DISKSIZE >/dev/null 2>&1
}
```

Networking

As mentioned above I want my VMs to look like real machines on the LAN the host server is connected on, this will require bridging the tap devices. Newer versions of qemu can automatically create and use tap device but I it will not do bridging and besides that it you need to tell it which tap device anyway. If you intend to run several VMs at once that all look like real servers on the LAN you will need to write qemu-ifup and qemu-ifdown scripts in /etc to deal with that.

I like to write a single script qemu-nethelper and have qemu-ifup and qemu-ifdown linked to it. The qemu-system-* emulators all execute /etc/qemu-ifup when bringing up a VM with a tap device, with the tap device parameter, and similarly execute /etc/qemu-ifdown when taking down a VM with a tap device.

As mentioned above newer versions of qemu (I think 1.1+) automatically create the tap device so the qemu-nethelper only needs to do the bridging. Now to make things a lot easier I like to have the host on which I run VMs with br0 configured at boot and then the qemu-nethelper only needs to add the

tap device to the bridge, making it extremely simple. If br0 is already configured at boot then you need not restart any iptables so long as the chains use the bridge devices and the kernel has support for ebtables.

```
#!/bin/bash
NAME=$(basename $0)
tun_up ()
{ /sbin/ifconfig $1 0.0.0.0 promisc up
  /usr/bin/sleep 0.5
  /sbin/brctl addif br0 $1
}

tun_down ()
{ /sbin/ifconfig $1 down
  /usr/bin/sleep 0.5
  /sbin/brctl delif br0 $1
}

case $NAME in
  qemu-ifup) tun_up $1;;
  qemu-ifdown) tun_down $1;;
  *) echo fail; exit 1;;
esac
```

Using Qemu from Unprivileged Users

Using root for doing your everyday tasks is commonly discouraged so let's see how we can work around using qemu from unprivileged users. Some say that it's sufficient to give sudo execution on /etc/qemi-if* but that not really enough because qemu-system-* needs to run as root or it will not be able to create the taps and access other resources. Although it is technically possible to give an unprivileged user sufficient privileges to execute correctly qemu-system-* emulators it is much easier to give users the sudo right to run qemu-system-* as privileged user.

```
User_Alias QEMUERS = al, john, jack

Cmnd_Alias QEMUCMD = /usr/bin/qemu-system-*

QEMUERS ALL=(ALL) NOPASSWD: QEMU
```

This would be sufficient to run the VMs as any of the unprivileged users in QEMUERS user alias (al, john, jack) but the management script would need to run sudo qemu-system-* …. this is easy to obtain:

```
[ $(/usr/bin/id -u) -ne 0 ] && CMD="sudo qemu-system-.... " || CMD="qemu-
system-...."
eval $(echo "$CMD &")
```

Alternatively you could give privileges to execute the management script as root.

Examples

Here are examples of the dialogs that user would see wile creating, starting, stopping and deleting a VM. q

VM Creation Example

```
+-----------------------------------------------+
| Choose Virtual Machine Name                   |
| N.B. The following names are already in use   |
| ORACLE                                        |
| ORACLE2                                       |
| armedslack                                    |
| freenas                                       |
| test                                          |
| +-------------------------------------------+ |
| | test2                                   | |
| +-------------------------------------------+ |
+-----------------------------------------------+
|           <  OK  >      <Cancel>              |
+-----------------------------------------------+
```

```
+--------------------------------+
| Choose the type of system to   |
| be emulated                    |
| +----------------------------+ |
| | (*) qemu-system-x86_64  1 | |
| | ( ) qemu-system-arm     2 | |
| | ( ) qemu-system-i386    3 | |
| +----------------------------+ |
+--------------------------------+
|     <  OK  >   <Cancel>        |
+--------------------------------+
```

```
+--------------------------------+
| Choose VM memory size in Mb    |
| (max 3421 Mb)                  |
| +----------------------------+ |
| |1024                        | |
| +----------------------------+ |
+--------------------------------+
|     <  OK  >   <Cancel>        |
+--------------------------------+
```

```
+-----------------------------+
| Choose VM Keyboard layout   |
| (cancel for default en-us)  |
```

```
| +--------------------------+ |
| |      (*) ar     1        | |
| |      ( ) de-ch  2        | |
| |      ( ) es     3        | |
| |      ( ) fo     4        | |
| |      ( ) fr-ca  5        | |
| |      ( ) hu     6        | |
| |      ( ) ja     7        | |
| |      ( ) mk     8        | |
| |      ( ) no     9        | |
| |      ( ) pt-br  10       | |
| |      ( ) sv     11       | |
| |      ( ) da     12       | |
| |      ( ) en-gb  13       | |
| |      ( ) et     14       | |
| |      ( ) fr     15       | |
| |      ( ) fr-ch  16       | |
| |      ( ) is     17       | |
| |      ( ) lt     18       | |
| |      ( ) nl     19       | |
| |      ( ) pl     20       | |
| |      ( ) ru     21       | |
| |      ( ) th     22       | |
| |      ( ) de     23       | |
| |      ( ) en-us  24       | |
| |      ( ) fi     25       | |
| |      ( ) fr-be  26       | |
| |      ( ) hr     27       | |
| |      ( ) it     28       | |
| |      ( ) lv     29       | |
| |      ( ) nl-be  30       | |
| |      ( ) pt     31       | |
| |      ( ) sl     32       | |
| |      ( ) t      33       | |
| +--------------------------+ |
+------------------------------+
|     <  OK  >   <Cancel>      |
+------------------------------+
```

```
+-----------------------------------------------+
| Choose installation image (Cancel for none)   |
| +-------------------------------------------+ |
| | (*) /ISO/CentOS-6.6-i386-bin-DVD1.iso   1 | |
| | ( ) /ISO/Fedora-Server-DVD-i386-21.iso  2 | |
| | ( ) /ISO/FreeNAS-9.10.1.iso             3 | |
| | ( ) /ISO/OL6.6-x86.iso                  4 | |
| | ( ) /ISO/rhel-server-6.6-i386-dvd.iso   5 | |
| +-------------------------------------------+ |
+-----------------------------------------------+
|           <  OK  >      <Cancel>              |
```

```
+---------------------------------------------+
```

```
+----------------------------+
| Choose VM kernel           |
| +------------------------+ |
| |                        | |
| +------------------------+ |
+----------------------------+
|   <  OK  >  <Cancel>       |
+----------------------------+
```

```
+----------------------------+
| Choose VM disk (cancel for |
| default)                   |
| +------------------------+ |
| |                        | |
| +------------------------+ |
+----------------------------+
|   <  OK  >  <Cancel>       |
+----------------------------+
```

```
+----------------------------+
| Choose VM disk image size  |
| +------------------------+ |
| |2G                      | |
| +------------------------+ |
|                            |
+----------------------------+
|   <  OK  >  <Cancel>       |
+----------------------------+
```

```
+----------------------------+
| Type VNC password for this |
| VM                         |
| +------------------------+ |
| |pleasechangethis        | |
| +------------------------+ |
+----------------------------+
|   <  OK  >  <Cancel>       |
+----------------------------+
```

```
+-------------Configuration for test2 VM-----------------+
| +-------------------------------------------------+ |
| |EMULATOR:   qemu-system-x86_64                   | |
| |MACHINE:    pc                                   | |
| |CPU:        host                                 | |
| |MEM:        1024                                 | |
| |KEYBOARD:   en-us                                | |
| |DISK:       /VM/test2/disk0.qcow2                | |
```

```
| |DISKSIZE:   2G                                          | |
| |CDROM:      /ISO/CentOS-6.6-i386-bin-DVD1.iso           | |
| |BIOS:       file=/usr/share/qemu/bios.bin               | |
| |BOOT_ORDER: c                                           | |
| |KVM:        1                                           | |
| |DISPLAY:    vnc                                         | |
| |VNCPW:      oleasechangethis                            | |
| |USB:        1                                           | |
| |KERNEL:                                                 | |
| |INITRD:                                                 | |
| |APPEND:                                                 | |
| |MAC:        52:54:57:c6:cc:21                           | |
| +-------------------------------------------------------+ |
|                                                           |
+-----------------------------------------------------------+
|                <  OK  >        <Cancel>                   |
+-----------------------------------------------------------+
```

Starting and Stopping VM Examples

```
+----------------------------+
| Choose Virtual Machine     |
| +------------------------+ |
| |   ( ) ORACLE      1    | |
| |   ( ) ORACLE2     2    | |
| |   ( ) armedslack  3    | |
| |   ( ) freenas     4    | |
| |   ( ) test        5    | |
| |   (*) test2       6    | |
| +------------------------+ |
+----------------------------+
|   <  OK  >  <Cancel>       |
+----------------------------+
```

After the command for starting the VM is executed 2 more things take place:

1. a quickstart.sh script (containing the required commands to start the VM) is created in the VM folder
2. the qemu-system-* command to start the VM is echoed back on the console

Stopping a virtual machine has the same dialog.

VM Delete Example

```
+----------------------------+
| Choose Virtual Machine     |
| +------------------------+ |
| |   ( ) ORACLE      1    | |
| |   ( ) ORACLE2     2    | |
```

```
| |   ( ) armedslack  3    | |
| |   ( ) freenas     4    | |
| |   ( ) test        5    | |
| |   (*) test2       6    | |
| +------------------------+ |
+----------------------------+
|   <  OK  >  <Cancel>       |
+----------------------------+
```

```
+-------Do you want to delete VM test2 ?---------+
| This will remove VM test2 with id:6 from the   |
| configuration. The folder containing the VM    |
| and the quiskstart.sh script will need to be   |
| manually removed.                              |
| Do you wish to proceed ?                       |
+------------------------------------------------+
|             < Yes >       < No  >              |
+------------------------------------------------+
```

```
+------NOTICE---------+
| Folder containing   |
| VM needs to be      |
| manually removed.   |
+---------------------+
```

You may be asking: “what is the quinskstart.sh script ?” Well each time the management script is used to start a VM it creates a quinskstart.sh (containing the required commands to start the VM) in the VM folder so that if the config database ever gets irreparably corrupt you can still start the VM with the quinskstart.sh.

Sources

I've a blog entry on LQ where I talk a little more extensively on minimizing the code in bash scripts.
Minimize the amount of code in your bash scripts

* Originally written by louigi600

howtos, louigi600