Xenapi Admin Tools 2.0 Scripting Guidelines

My XCP scripts have evolved over the few months I've been writing them. I kept running into the same problems in each script so I created some standard functions and a pretty standard work flow for them all. As of now (September 18th 2012) all of my non-interactive scripts have been converted to this new format. Following is an explanation of each standard section. Depending on the problems I'm trying to solve I sometimes need to change the solutions in a small way. However, having said that it's been amazing how easy it has been to add features to scripts that didn't have them. Following is a list.

Xenapi Admin Tools 2.0 features

1. All scripts can display objects by UUID
2. All scripts can display objects as name-label
3. All scripts can display data in tabulated text format
4. All scripts can display data in Comma Separated Values format
5. All scripts should display the script name and version in the help
6. All scripts should include the changelog at the beginning in comments
7. All scripts should be able to present data in color (cecho and setcolors functions)
8. All scripts should only loop UUID lists, never name-labels
9. All scripts hold data in arrays that can be passed to getcolwidth
10. All scripts should display column padding using printspaces if possible (built in CSV support)
11. All scripts should present name-label lists in alphabetical order (sort_names function)
12. All scripts should take the column padding size as an argument

Xenapi Admin Tools 2.0 Sections

Below I've used the lsvms.sh script to outline the sections of my scripts.

Section . – Interpreter line

```
#!/bin/bash
```

Section 2 – Changelog

```
# Lists XCP/Xenserver Virtual Machines one per line with uuid
and host
# Author: Grant McWilliams (grantmcwilliams.com)
# Version: 0.5
# Date: June 27, 2012
# Version: 0.6
# Date: Sept 15, 2012
# Complete rewrite using printspaces, sort_vmnames and
getcolwidth
# Now provides four output MODES - name, uuid. mixed and both
# Also provides CSV output
```

Section 3 – Standard functions – setup(), cecho(), setcolors(), Syntax()

The setup() function sets initial variables and executes setcolors. Setup is always called before getopts in case you want a default variable to be set that may be overridden by a user inputted option on the commandline which is processed by getopts.

```
setup()
{
     setcolors
     IFS=$'\n'
     MINSPACE="5"
     MODE="mixed"
     VERSION="0.7"
}
```

The setcolors() function assigns ansi values to variables. The cecho() function below uses these variables to echo messages in color. To set a color use `cecho "Hello World in Blue"` blue. To turn off the ansi values use the off keyword eg. `cecho "Hello world" off`. The cecho function will display a quoted message in the color specified. It does not print a carriage return so must be followed by an `echo ""` if one is needed.

```
#give names to ansi sequences
setcolors()
{
     black='\e[30;47m'
     white='\e[37;47m'
     red='\e[0;31m'
     blue='\e[0;34m'
     cyan='\e[0;36m'
     off='\e[0m'
}

#color echo
cecho ()
{
     MSG="${1}"
     if [ -z $2 ] ;then
          color="white"
     else
          eval color=\$$2
     fi
     echo -ne "${color}"
     echo -ne "$MSG"
     echo -ne "${off}"
}
```

The syntax() function gets called from getopts if the option provided by the user is not recognized or is -h. Syntax should display the tool version and it's usage.

```
syntax()
{
     echo “$(basename $0) $VERSION”
     echo ""
     echo "Syntax: $(basename $0) [options]"
     echo "Options:"
     echo "-d - shell debugging"
     echo "-c - output comma seperated values"
     echo "-u - shows VM UUID, Status, Host UUID"
     echo "-b - shows VM Name, Status, VMUUID, Host Name and Host UUID"
     echo "-n - shows VM Name, Status and Hostname"
     echo "-m - shows VM Name, Status, VM UUID and Hostname"
     echo "-h - this help text"
     echo ""
     exit
}
```

Section 4 – Xenapi Admin Tools 2.0 functions – printspaces(), sort_names() and getcolwidth().

Xenapi Admin Tools 2.0 functions are more complex functions solving problems very specific to the goal of Xenapi Admin Tools – displaying lists of information. They're very useful and reusable for the most part.

The getcolwidth() function takes an array as an argument and returns an integer representing the length of the longest item line the array. For instance if you had an array that had these items

```
array[0]=bob
array[1]=sally
array[2]=rumplestilskin
```

and you wanted to know how long the longest item is (array[2] you'd pass the entire thing to getcolwidth like this.

```
COLLONGEST[0]=$(getcolwidth "${TITLES[2]}" "${VMUUIDS[@]}")
```

The getcolwidth() function calculates the longest item in TITLES[2] and VMUUIDS[@] and returns an integer which is assigned to COLLONGEST[0]. This way I have the longest item in column one in COLLONGEST[0], the longest item in column two in COLLONGEST[1] and so on.

```
#get width of columns
getcolwidth()
{
        #get longest item in array and returns an integer
        array=( "$@" )
        i=0
        LONGEST="0"
        IFS=$'\n'
        for ITEM in ${array[@]} ;do
```

```
            if [[ "${#ITEM}" -gt "$LONGEST" ]] ;then
                 LONGEST="${#ITEM}"
            fi
       done
       echo "$LONGEST"
  }
```

The printspaces() function takes two arguments - the longest item in a column returned from getcolwidth() and the length of the current item returned from bash. To check the length of $VAR use the following syntax -${#VAR}.

```
  printspaces()
  {
       # arg 1 - the longest item in the column
       # arg 2 - the length of the item ie. ${#VAR} (integer)
       COLUMN="$1"
       ITEM="$2"

       if [[ "$CSV" = "yes" ]] ;then
            echo -ne ","
       else
            printf "%*s" "$(( $COLUMN + $MINSPACE - $ITEM ))"
       fi
  }
```

The sort_names() function needs modification depending on the needs of the script. It sorts one array for each move it also moves items of another array. A good example would be an array that holds VM UUID's with a similar array holding VM name-labels. If you want to sort the name-labels alphabetically and also want the array holding the UUIDs to still match up you need to move the UUID array items at the same time. Only use on arrays that are "synchronized" ie. items that correlate based on the array index. In the example below the array VMNAMES is looped through and each two lines are bubble sorted. As the items in VMNAMES are moved so are the correlating items in VMUUIDS, STATES, HOSTUUIDS and HOSTNAMES. After sort_vmnames exits all of these arrays are still correlated but VMNAMES is now alphabetical.

```
  sort_vmnames()
  {
       local MAX=$((${#VMNAMES[@]} - 1))
      while ((MAX > 0)) ;do
            local i=0
          while ((i < MAX)) ;do
                 local j=$((i + 1))
                 if expr "${VMNAMES[$i]}" \> "${VMNAMES[$j]}"
  >/dev/null
                 then
```

```
                        local t=${VMNAMES[$i]}
                        local u=${VMUUIDS[$i]}
                        local v=${STATES[$i]}
                        local w=${HOSTUUIDS[$i]}
                        local x=${HOSTNAMES[$i]}
                        VMNAMES[$i]=${VMNAMES[$j]}
                        VMUUIDS[$i]=${VMUUIDS[$j]}
                        STATES[$i]=${STATES[$j]}
                        HOSTUUIDS[$i]=${HOSTUUIDS[$j]}
                        HOSTNAMES[$i]=${HOSTNAMES[$j]}
                        VMNAMES[$j]=$t
                        VMUUIDS[$j]=$u
                        STATES[$j]=$v
                        HOSTUUIDS[$j]=$w
                        HOSTNAMES[$j]=$x
                    fi
                    ((i++))
                done
                ((MAX--))
            done
        }
```

Section 5 – setup and get options.

The first item of the main program codebase to run is the setup() function followed by getopts processing the commandline options.

```
setup
while getopts :dcubnmh opt ;do
        case $opt in
                d) set -x ;;
                h) syntax ;;
                c) CSV="yes" ;;
                u) MODE="uuid" ;;
                b) MODE="both" ;;
                n) MODE="name" ;;
                m) MODE="mixed" ;;
                \?) echo "Unknown option"; syntax ;;
        esac
done
shift $(($OPTIND - 1))
```

Section 6 – Populating the TITLE[] array

We use an array to store all column titles. We use a case statement to assign different values depending on the commandline options. This is a very flexible system.

```
# Set Title array depending on MODE
case "$MODE" in
     "uuid") TITLES=( 'VM UUID' 'Status' 'Host UUID' ) ;;
     "name") TITLES=( 'VM Name' 'Status' 'Host Name' ) ;;
     "mixed")  TITLES=( 'VM Name' 'Status' 'VM UUID' 'Host Name'
) ;;
     "both") TITLES=( 'VM Name' 'Status' 'VM UUID' 'Host Name'
'Host UUID' ) ;;
esac
```

Section 7 – Populating the data arrays

We populate the data arrays her in order to pass them to getcolwidth which returns the longest item in the array and in turn populates COLLONGEST[]. It's best to synchronize the data if possible. For instance we're looping through the VM UUIDs (always loop UUIDs, not name-labels) and for each one we retrieve the VM name-label and assigne it to VMNAMES. VMUUIDS[0] and VMNAMES[0] are referencing the same VM. This way we can sort them using sort_names() and keep the correlation.

It may seem redundant to populate these arrays and then later in the main loop get the data again and sometimes it is. Sometimes you can just reference THESE array items and you don't have to gather the data again in the main loop which makes the main loop code very clean.

```
# Populate arrays for VM UUIDs, VM name-label, power state, Host
Name and Host UUID

VMUUIDS=( $(xe vm-list params=uuid is-control-domain=false
--minimal | sed 's/,/\n/g') )
for i in $(seq 0 $(( ${#VMUUIDS[@]} - 1 )) ) ;do
     VMNAMES[$i]=$(xe vm-param-get uuid="${VMUUIDS[$i]}" \
     param-name=name-label)
     STATES[$i]=$(xe vm-param-get uuid="${VMUUIDS[$i]}"  \
     param-name=power-state)
     HOSTUUIDS[$i]=$(xe vm-param-get uuid="${VMUUIDS[$i]}" \
     param-name=resident-on)
     if [[ "${HOSTUUIDS[$i]}" = '<not in database>' ]] ;then
          HOSTUUIDS[$i]="" ; HOSTNAMES[$i]=""
     else
          HOSTNAMES[$i]=$(xe host-param-get \
          uuid="${HOSTUUIDS[$i]}" param-name=name-label)
     fi
done
```

Section 8 – Sort the data arrays

For presentation purposes we want the data arrays sorted alphabetically according to name-label. We use the sort_names() function. For each different application sort_names has to be changed slightly to match what is being sorted. NOTE: if you're sorting a LOT of data you may be better off looping through it and piping the loop to sort and saving the output.

```
# Bubble sort the vm date in alphabetical order by VMNAMES[]
sort_vmnames
```

Section 9 – Get the length of the items in the data arrays

Once again we use a case statement to get the length of each column depending on which commandline option the user chose. This is a very flexible way of switching between using UUIDs or Name-labels for display.

```
# Get the length of each column and store it in COLLONGEST[]
case "$MODE" in
     "uuid") COLLONGEST[0]=$(getcolwidth "${TITLES[2]}" "$
{VMUUIDS[@]}")
               COLLONGEST[1]=$(getcolwidth "${TITLES[1]}" "$
{STATES[@]}")
               COLLONGEST[2]=$(getcolwidth "${TITLES[4]}" "$
{HOSTUUIDS[@]}")
      ;;
     "name") COLLONGEST[0]=$(getcolwidth "${TITLES[0]}" "$
{VMNAMES[@]}")
               COLLONGEST[1]=$(getcolwidth "${TITLES[1]}" "$
{STATES[@]}")
               COLLONGEST[2]=$(getcolwidth "${TITLES[3]}" "$
{HOSTNAMES[@]}")
      ;;
     "mixed") COLLONGEST[0]=$(getcolwidth "${TITLES[0]}" "$
{VMNAMES[@]}")
               COLLONGEST[1]=$(getcolwidth "${TITLES[1]}" "$
{STATES[@]}")
               COLLONGEST[2]=$(getcolwidth "${TITLES[2]}" "$
{VMUUIDS[@]}")
               COLLONGEST[3]=$(getcolwidth "${TITLES[3]}" "$
{HOSTNAMES[@]}")
      ;;
      "both") COLLONGEST[0]=$(getcolwidth "${TITLES[0]}" "$
{VMNAMES[@]}")
               COLLONGEST[1]=$(getcolwidth "${TITLES[1]}" "$
{STATES[@]}")
               COLLONGEST[2]=$(getcolwidth "${TITLES[2]}" "$
{VMUUIDS[@]}")
               COLLONGEST[3]=$(getcolwidth "${TITLES[3]}" "$
{HOSTNAMES[@]}")
               COLLONGEST[4]=$(getcolwidth "${TITLES[4]}" "$
{HOSTUUIDS[@]}")
      ;;
esac
```

Section 10 – Print the column headings

Looping through the TITLES[] array is a very easy way to print the column headings. Keep in mind that this array is designed to be printed horizontally on the screeen ie. without carriage returns.

```
# Print column headings
```

```
     for i in $(seq 0 $(( ${#TITLES[@]} - 1 )) ) ;do
          cecho "${TITLES[$i]}" off
          printspaces "${COLLONGEST[$i]}"  "${#TITLES[$i]}"
     done
     echo ""
```

Section 11 – Printing the data columns

The last section outputs the data columns to the user. We loop through the VM UUIDS (again never loop through name-labels) and using a case statement print out data differently depending on the options selected on the commmandline. We use cecho to print in color and printspaces to print out the padding between columns. Because we populated the arrays in section 7 we don't have to do anything but present data here. This is not always 100% possible though. On occasion you may also have to gather data in this loop.

```
# Print data columns
for i in $(seq 0 $(( ${#VMUUIDS[@]} - 1 )) ) ;do
     case "$MODE" in
          "uuid")
               cecho "${VMUUIDS[$i]}" cyan
               printspaces "${COLLONGEST[0]}" "${#VMUUIDS[$i]}"
               cecho "${STATES[$i]}" red
               printspaces "${COLLONGEST[1]}" "${#STATES[$i]}"
               cecho "${HOSTUUIDS[$i]}" blue
               printspaces "${COLLONGEST[3]}" "${#HOSTUUIDS[$i]}"
          ;;
          "name")
               cecho "${VMNAMES[$i]}" cyan
               printspaces "${COLLONGEST[0]}" "${#VMNAMES[$i]}"
               cecho "${STATES[$i]}" red
               printspaces "${COLLONGEST[1]}" "${#STATES[$i]}"
               cecho "${HOSTNAMES[$i]}" blue
               printspaces "${COLLONGEST[3]}" "${#HOSTNAMES[$i]}"
          ;;
          "mixed")
               cecho "${VMNAMES[$i]}" cyan
               printspaces "${COLLONGEST[0]}" "${#VMNAMES[$i]}"
               cecho "${STATES[$i]}" red
               printspaces "${COLLONGEST[1]}" "${#STATES[$i]}"
               cecho "${VMUUIDS[$i]}" blue
               printspaces "${COLLONGEST[2]}" "${#VMUUIDS[$i]}"
               cecho "${HOSTNAMES[$i]}" blue
               printspaces "${COLLONGEST[3]}" "${#HOSTNAMES[$i]}"
          ;;
          "both")
               cecho "${VMNAMES[$i]}" cyan
               printspaces "${COLLONGEST[0]}" "${#VMNAMES[$i]}"
               cecho "${STATES[$i]}" red
               printspaces "${COLLONGEST[1]}" "${#STATES[$i]}"
               cecho "${VMUUIDS[$i]}" blue
```

```
                    printspaces "${COLLONGEST[2]}" "${#VMUUIDS[$i]}"
                    cecho "${HOSTNAMES[$i]}" blue
                    printspaces "${COLLONGEST[3]}" "${#HOSTNAMES[$i]}"
                    cecho "${HOSTUUIDS[$i]}" blue
                    printspaces "${COLLONGEST[4]}" "${#HOSTUUIDS[$i]}"
                ;;
        esac
        echo ""
    done
```

Xenapi Admin Tools 3.0 proposed features

1. All scripts should work with xe remote ie. By specifying the poolmaster username and ip address the scripts could be run remotely
2. All scripts will use generic functions from xcptools_library.sh
3. A function needs to be written to take the sorted UUID/Name-label pairs and create a select menu from it. Depending on the users selection the function will return the UUID portion only.
4. All listing scripts (lsvms, lshosts, lsvdis etc.) should offer a -s <sort field> option. This relies on item 3 above.