

Xenapi Admin Tools 3.0 Scripting Guidelines

All Xenapi Admin Tools 3.0 scripts should include the following functionality

Xenapi Admin Tools 3.0 scripts should

1. display objects by UUID or Name-labels
2. display data in tabulated text format or CSV
3. handle having more than one set of column titles
4. know which directory it resides and source library.sh
5. use generic functions from library.sh
6. only loop through UUID lists, never Name-labels
7. hold data in arrays that can be passed to getcolwidth function
8. display the script name and version in the help
9. include the changelog at the beginning in comments
10. offer a “sort by field” option using -s (using sort_array function)
11. show UUID/Name pairs sorted by name in select menu (show_arraymenu function)
12. be able to present data in color (cecho and setcolors functions)
13. display column padding using printspaces if possible (built in CSV support)
14. present lists (Name-label | UUID) in alphabetical order (sort_array function)
15. take column padding as a commandline argument
16. operate on remote poolmasters by specifying username/ip or load stored local config (get_poolcreds function)

Xenapi Admin Tools 3.0 Sections

Below I've used the lsvms.sh script as an example of XCP tools 3.0.

Section 1 – 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 2 – Standard functions – setup(), 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()
{
    SCRIPTDIR=$(dirname $(readlink -f "${BASH_SOURCE[0]}"))
    source "$SCRIPTDIR/library.sh"
```

```

setcolors
DEFSPACE="5"
MINSPEACE="$DEFSPACE"
MODE="mixed"
VERSION="0.7"
}

```

The syntax() function gets called from getopt() 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 "-o <value> - changes sort order by column, value can be vmname, hostname, vmuuid, hostuud"
    echo "-s <host> - remote poolmaster host"
    echo "-p <password> - remote poolmaster password"
    echo "-h - this help text"
    echo "-w - number of whitespaces between columns"
    echo ""
    exit
}

```

Section 3 – run setup and get options.

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

```

setup
while getopt :dcubnmhw:s:o:p: opt ;do
    case $opt in
        d) set -x ;;
        h) syntax ;;
        c) CSV="yes" ;;
        u) MODE="uuid" ;;
        b) MODE="both" ;;
        n) MODE="name" ;;
        m) MODE="mixed" ;;
        o) ORDER="$OPTARG" ;;
        s) REMOTE="yes" ; POOLMASTER="$OPTARG" ;;
        p) PASSWORD="$OPTARG" ;;
        w) isnumber "$OPTARG" && MINSPEACE="$OPTARG" ;;
        \?) echo "Unknown option"; syntax ;;
    esac
done

```

```

        esac
    done
    shift $(( $OPTIND - 1 ))
    getpoolcreds

```

Section 4 – 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 5 – Populating the data arrays

We populate the data arrays in order to pass them to getcolwidth which returns the longest item in the array and in turn populates COLLONGEST[]. It's imperative that the data arrays are synchronized ie.

```

VMUUIDS[1]="8ea244fd-b67a-78c2-9317-18adb29ae8cc"
VMUUIDS[2]="4303c7e3-0479-42bc-c115-5a3877f1403f"

VMNAMES[1]="webserver"
VMNAMES[2]="mailserver"

```

This is the best way to do associative arrays in Bash 2 or 3. This method does not allow sparse arrays so make sure every array item has a value even if it's blank. Below we're populating the data arrays for VM uuid, VM name-label, power-state, Host name and Host uuid.

```

# 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 6 – Sort the data arrays

For presentation purposes we want the data arrays sorted alphabetically according to name-label. We

use the `sort_arrays()` function for this.

```
case "$ORDER" in
    vmname) sort_arrays VMNAMES VMUUIDS STATES HOSTUUIDS HOSTNAMES ;;
    hostname) sort_arrays HOSTNAMES VMNAMES VMUUIDS STATES HOSTUUIDS ;;
    vmuuid) sort_arrays VMUUIDS HOSTUUIDS HOSTNAMES VMNAMES STATES ;;
    hostuuid) sort_arrays HOSTUUIDS HOSTNAMES VMNAMES VMUUIDS STATES ;;
esac
```

Section 7 – Get the length of the items in the data arrays

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 8 – Printing the column headings and data columns

The last section outputs the column headings and data columns to the user. We loop through the VM UUIDS (never loop through name-labels) and using a case statement to print out data differently depending on the options specified by the user. We use `cecho` to print in color and `printf` 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 column headings and data columns
printheadings
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[$i]}" blue
    ;;
    "name")
        cecho "${VMNAMES[$i]}" cyan
        printspaces "${COLLONGEST[0]}" "${#VMNAMES[$i]}"
        cecho "${STATES[$i]}" red
        printspaces "${COLLONGEST[1]}" "${#STATES[$i]}"
        cecho "${HOSTNAMES[$i]}" blue
    ;;
    "mixed")
        cecho "${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
    ;;
    "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
    ;;
esac
echo ""
done

```

Xenapi Admin Tools 3.0 Functions in Depth

setcolors() and cecho()

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.

```
setcolors
```

```
cecho "Warning!!" red
```

getcolwidth()

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]=rumpelstilskin
```

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.

printspaces()

The `printspaces()` function takes two arguments - the longest item in a column returned from `getcolwidth()` and the length of the current item. To check the length of `$VAR` use the following syntax `${#VAR}`. Below we're echoing `${VMNAME[1]}`, then passing `${COLLONGEST[1]}` and `${#VMNAME[1]}` to `printspaces()`. `Printspaces()` will subtract the length of `${VMNAME[1]}` from `${COLLONGEST[1]}` and print that amount of empty spaces for proper column padding.

```
cecho "${VMNAME[1]}" red ; printspaces "${COLLONGEST[1]}" "${#VMNAME[1]}"
```

printheadings()

The `printheadings()` function prints out the column headings previously assigned in `TITLES[]`.

Just call it with no arguments.

```
printheadings
```

sort_arrays()

The `sort_arrays()` function sorts multiple arrays in parallel. The

first array passed to it is used to determine the order of all array items. Whatever is done to the first array will be done to the others. For example you have two arrays indexed together.

```
VMUUUIDS[1]="8ea244fd-b67a-78c2-9317-18adb29ae8cc"  
VMUUUIDS[2]="4303c7e3-0479-42bc-c115-5a3877f1403f"
```

```
VMNAMES[1]="webserver"  
VMNAMES[2]="mailserver"
```

In this case the UUID for "webserver" is 8ea244fd-b67a-78c2-9317-18adb29ae8cc because the array index for both is 1 (VMUUUIDS[1] and VMNAMES[1]). We have to have multiple arrays indexed together with Bash 2 and 3 because they don't support associative arrays. To sort both of these arrays and keep the associations correct we'd use the `sort_arrays()` function. Note: you specify the array NAME only eg. VMNAMES not \$VMNAMES or VMNAMES[@].

```
sort_arrays VMNAMES VMUUUIDS
```

fsort_arrays()

The `fsort_arrays()` function works just like `sort_arrays` and even uses the same syntax. The difference is that it prints out the value of the arrays passed into a temporary file and then uses the `sort` command to sort it. This should be faster for very large arrays but requires there to be a `TMPDIR` with enough room for the textfile. Make sure `TMPDIR` is created in setup, a trap is created to call `cleanup()` when the user presses control-c and a `cleanup()` function is created to delete the `TMPDIR`.

```
setup  
....  
sort_arrays VMNAMES VMUUUIDS  
....  
cleanup
```

yesno()

The `yesno()` function just asks a simple question and returns a 0 or 1 depending on the answer. `yesno()` takes one argument – the message to display.

To use `yesno()` you just place it in an if conditional followed by the message to display.

```
If yesno "Do you like dogs?" ; then  
    echo "you like dogs"  
else  
    echo "You don't like dogs"  
fi
```

isnumber()

The `isnumber()` function checks to see if the value in the variable only contains an integer. If any other characters are found it will return 1.

```
If isnumber "43" ; then
    echo "is a number"
else
    echo "is not a number"
fi
```

getunit()

The `getunit()` function will convert an integer of bytes to Kilobytes, Megabytes, Gigabytes or Terabytes. It returns the number followed by an abbreviation ie. 964M or 123G.

To use it just pass the integer to the `getunit()` function.

```
SIZE=$(getunit 9456)
```

getpoolcreds()

The `getpoolcreds()` function checks for the existence of `$SCRIPTDIR/.XECONFIG`. The `.XECONFIG` directory holds one config file per pool. Inside each config file are lines for poolmaster, username, port and password.

```
POOLMASTER="cloud0"
PORT="443"
USERNAME="root"
PASSWORD="password"
```

The permissions are checked on the `.XECONFIG` directory and if they're readable by anyone else besides root the script throws an error and exits. This behavior may change in the future and offer the user the chance of modifying the permissions.

If `.XECONFIG` exists `getpoolcreds()` parses each text file who's name ends in `.cfg` for hostname, port, username and password. If any items are not set it prompts the user for the value. The variable `XE_EXTRA_ARGS` is set and exported globally. If no configs are found it assumes the poolmaster is localhost.

Using `getpoolcreds` is as simple as just calling it with no arguments.

```
getpoolcreds
```

show_arraymenu()

The `show_arraymenu()` function is a semi-hard coded function that takes at least one array as an argument and presents a select menu to the user. When the user selects an item `show_arraymenu()` strips any valid UUID from that selection and returns it.

The purpose of such a function would be to present the user with a menu of Name-label and UUID pairs, when the user selects a Name-label (UUID is for when there are multiple identical name-labels) the UUID is returned. It allows a very easy way to present the user with a sorted list of name-labels and then return a reliable UUID for further processing.

Pass at least one array when calling `show_arraymenu()`

```
show_arraymenu VMNAMES VMUUIDS
```

Note 1: only pass the array names not the arrays themselves ie. `VMNAMES` not `$VMNAMES` or `${VMNAMES[@]}`.

Note 2: do NOT pass more than one UUID array to `show_arraymenu` as it will attempt to strip ANY UUIDs and return them. It only handles one UUID.