zertifix

```
#!/usr/bin/env bash

# Script for: * Generating a list of iLOs and to save it as a file.
#             * Work on that list of iLOs to generate CSRs
#               and to save the CSRs to a filesystem.
#
# This script should have been able to do a lot more than it is able to do yet,
# but unfortunately there was not enough time and a lot of things went wrong.
#
# Please keep in mind that ACME has to get a lot of things together before this
# script will go anywhere...
#     * Order a new cluster of Insta Certifier appliances
#     * Set them up
#     * Get a license for the REST-API of Insta Certifier
#     * and so on...


# Debug option. Uncomment to activate
# set -x;


# ----------------------------------------------------
# SETUP THE SCRIPT
# ----------------------------------------------------

# Include the configuration file if it can be found
if [[ -e ../etc/zertifix_conf.sh ]];
then
    source ../etc/zertifix_conf.sh;
else
    echo "Configuration file not found!";
    exit 1;
fi

# Include functions
if [[ -e $SCRIPT_LIB/zertifix_include_lib.sh ]];
then
    source "${SCRIPT_LIB}/zertifix_include_lib.sh";
else
        echo "Library not found!";
    exit 1;
fi


# ----------------------------------------------------
# PARSE THE ARGUMENTS PASSED TO THE SCRIPT
# ----------------------------------------------------

f_parse_args "$@";

# Debug option
f_echo_args "$@";



# ----------------------------------------------------
# RUN THE SCRIPT
# ----------------------------------------------------

f_run_controller;

exit 0;
```

```
#!/usr/bin/env bash
# Config for zertifix

# ------------------------------------------------------------------------------
# CONFIGURATION VARIABLES
# ------------------------------------------------------------------------------

# Full path to zertifix script location
SCRIPT_HOME="/home/nq10004071/scripts/zertifix";

# Name of the file which will contain the FQDNs of the iLOs
ILO_FQDN_LIST_NAME="ilo_fqdn.lst";

# FQDN of the Red Hat Satellite server
SATELLITE_FQDN="hostname.lala.lala.domain.com";

# Name of the directory (in the scripts var-directory) which contains
# the hostlists and the backups of hostlists
DIR_HOSTLISTS_NAME="hostlists";

# Number of days to keep hostlists (retention time)
HOSTLISTS_RETENTION_TIME="186";

# The username and the password which will be used by hpilo_cli to
# connect to the iLOs
ILO_USER="Administrator";
ILO_PASS="**************";

# Name of the directory (in the scripts var-directory) which contains
# the directory structure for storing the CSRs
DIR_CSRS_NAME="CSRs";

# CSR fields
CSR_COUNTRY="DE";
CSR_STATE="Bayern";
CSR_LOCALITY="Munich";
CSR_ORGANIZATION="ACME";
CSR_ORGANIZATIONAL_UNIT="ST";

# Name of the directory, below script-var, containing the todo list files
DIR_TODO_LIST_DIR_NAME="TODO";


# ------------------------------------------------------------------------------
# DO NOT EDIT BELOW HERE - EXCEPT YOU KNOW WHAT YOU ARE DOING
# ------------------------------------------------------------------------------

# Build etc-path for the script
SCRIPT_ETC="${SCRIPT_HOME}/etc";
# Build lib-path for the script
SCRIPT_LIB="${SCRIPT_HOME}/lib";
# Build var-path for the script
SCRIPT_VAR="${SCRIPT_HOME}/var";
# Build bin-path for the script
SCRIPT_BIN="${SCRIPT_HOME}/bin";
# Build full path to the directory containing the lists of iLO-FQDNs
DIR_ILO_FQDN_LISTS_PATH="${SCRIPT_VAR}/${DIR_HOSTLISTS_NAME}";
# Build full path to the file containing the list of iLO-FQDNs
ILO_FQDN_LIST_FULL_PATH="${DIR_ILO_FQDN_LISTS_PATH}/${ILO_FQDN_LIST_NAME}";
# Build the path to the directory containing the directory structure
# which will contain the CSRs
CSRS_ROOT_DIR="${SCRIPT_VAR}/${DIR_CSRS_NAME}";
# Build the path to the directory containing the files for the todo list
# which will contain the todo list files
DIR_TODO_LIST_DIR_PATH="${SCRIPT_VAR}/${DIR_TODO_LIST_DIR_NAME}";
```

```
#!/usr/bin/env bash
```

```
# see https://github.com/mattbryson/bash-arg-parse/blob/master/arg_parse_example
# https://github.com/mattbryson/bash-arg-parse/blob/master/LICENSE
# MIT License (MIT)

# ------------------------------------------------------------------------------
# Function:    f_usage
#
# Description: This function prints short instructions for usage of the script.
#
# Arguments:   none
#
# Output:      none
#
# Return code: none
# ------------------------------------------------------------------------------

function f_usage ()
{
    echo "   $(tput setab 1)$(tput setaf 7)$(tput bold)";
    echo "
";
    echo "   ! ATTENTION ! ATTENTION ! ATTENTION ! ATTENTION ! ATTENTION ! ATTENTION ! ATTENTION !
ATTENTION !   ";
    echo "
";
    echo "
";
    echo "   zertifix requires the package python-hpilo-3.9-6.el7.noarch : iLO automation from python or
shell   ";
    echo "   to work!
";
    echo "
";
    echo "
";
    echo "   ! ATTENTION ! ATTENTION ! ATTENTION ! ATTENTION ! ATTENTION ! ATTENTION ! ATTENTION !
ATTENTION !   ";
    echo "
$(tput sgr0)";
    echo "   ";
    echo "   ";
    echo "   ";
    echo "   $(tput bold)Description:$(tput sgr0)";
    echo "   $(tput bold)------------$(tput sgr0)";
    echo "   ";
    echo "   At the moment zertifix is able to manage the following things:";
    echo "       - create a list with FQDNs of iLOs";
    echo "       - create CSRs on the iLOS and collect them";
    echo "   ";
    echo "   ";
    echo "   ";
    echo "   $(tput bold)usage:$(tput sgr0) zertifix -t task -m mode | -h | <positional_argument1
positional_argument2 ...] ";
    echo "   $(tput bold)------$(tput sgr0)";
    echo "   ";
    echo "   ";
    echo "     -t | --task            : Desired task <hostlist|csr>";
    echo "   ";
    echo "     -m | --mode            : Desired mode:";
    echo "                              * the task hostlist has two modes: <generate|backup>";
    echo "                                   * generate   : generates a list with FQDNs of iLOs.";
    echo "                                   * backup     : creates a backup of the file containing";
    echo "                                                  the list with FQDNs of iLOs.";
    echo "   ";
    echo "                              * the task csr has two modes: <generate|backup>";
    echo "                                   * generate   : generates the CSRs and puts them into";
    echo "                                                  the directory structure under var/CSRs";
    echo "                                   * todo       : creates a backup of the file containing";
    echo "                                                  the list with FQDNs of iLOs.";
    echo "   ";
```

```
    echo "     -h | --help            : Prints this message";
    echo "   ";
    echo "   ";
    echo "   ";
    echo "   ";
    echo "   $(tput bold)Usage examples:$(tput sgr0)";
    echo "   $(tput bold)---------------$(tput sgr0)";
    echo "   ";
    echo "       To print the help:";
    echo "       zertifix --help";
    echo "   ";
    echo "       To generate a list with iLOs:";
    echo "       zertifix --task hostlist --mode generate";
    echo "   ";
    exit 1;
}


# -----------------------------------------------------------------------------
# Function:             f_parse_args
#
# Description:          This function parses the named arguments passed to the script
#                       and puts them into variables. It also takes the positional
#                       arguments and puts them into an array "args".
#
# Named arguments:      -d, --debug: Debug mode
#                       -t, --task: The task
#                       -m, --mode: The mode
#                       -p, --paramaters: The parameters
#
# Positional arguments: 1: I do not know yet #TODO
#                       2: I do not know yet #TODO
#                       3: I do not know yet #TODO
#
# Output:      none
#
# Return code: none
# -----------------------------------------------------------------------------

function f_parse_args ()
{
  # positional args
  args=()

  # named args
  while [ "$1" != "" ]; do
      case "$1" in
          -d | --debug)       debug="$2";      shift;;
          -t | --task )       task="$2";       shift;;
          -m | --mode )       mode="$2";       shift;;
          -p | --parameters ) parameters="$2"; shift;;
          -h | --help )       f_usage;         exit;; # quit and show usage
          * )                 args+=("$1")            # if no match, add it to the positional args
      esac
      shift # move to next kv pair
  done

  # restore positional args
  set -- "${args[@]}"

  # set positionals to vars
#  positional_1="${args[0]}"
#  positional_2="${args[1]}"
#  positional_3="${args[2]}"



  # validate required args
  if [[ -z "${task}" || -z "${mode}" ]]; then
      # TODO
```

```
      echo "Invalid arguments"
      f_usage
      exit 1;
  fi

  # set defaults
  if [[ -z "$debug" ]]; then
      debug="no";
  fi
}



# ------------------------------------------------------------------------------
# Function:    f_echo_args
#
# Description: This function prints the arguments parsed by this script in
#              human readable form for debugging purposes.
#
# Arguments:   none
#
# Output:      String - The arguments
#
# Return code: none
# ------------------------------------------------------------------------------

function f_echo_args ()
{
  if [ "$debug" == "yes" ];
  then
    local counter="1";
    echo -e "\n\nThe arguments have been parsed as:\n";
    echo "named arg: debug:      $debug";
    echo "named arg: task:       $task";
    echo "named arg: mode:       $mode";
    echo "named arg: parameters: $parameters";
    for positional_arg in "${args[@]}"
    do
      echo "positional argument ${counter}: ${positional_arg}";
      let "counter++";
    done
fi
}
```

```
#!/usr/bin/env bash

# CONTROLLER
function f_run_controller ()
{
    case "$task" in
    "hostlist")
                echo "task: ${task}";
                case "$mode" in
                "generate")
                        # generate a list containing FQDNs of iLOs
                        echo "mode: ${mode}";
                        f_hostlist_generate;
                        ;;
                "backup")
                        # backup the list containing FQDNs of iLOs
                        echo "mode: ${mode}";
                        f_hostlist_backup;
                        ;;
                *)
            echo "mode: ${mode}";
                        echo "something went wrong!";
                        exit 1;
                        ;;
```

```
                esac
                ;;
    "csr")
        echo "task: ${task}";
        case "$mode" in
        "generate")
            echo "mode: ${mode}";
                        # create the directory structure to contain the CSRs
                        # and fetch the CSRs
            f_csr_generate;
            ;;
        *)
            echo "mode: ${mode}";
            echo "something went wrong!";
            exit 1;
            ;;
        esac
        ;;
    "help")
        echo "task: ${task}";
                f_print_usage;
                exit 1;
                ;;
    *)
        echo "task: ${task}";
        echo "something went wrong!";
                echo "";
                f_print_usage;
        exit 1;
        ;;
    esac
}
```

```
#!/usr/bin/env bash

# ------------------------------------------------------------------------------
# GLOBAL VARIABLES
# ------------------------------------------------------------------------------

# initialize the array A_ILO_HOSTNAMES
A_ILO_HOSTNAMES="";

# ------------------------------------------------------------------------------
# FUNCTIONS
# ------------------------------------------------------------------------------

# ------------------------------------------------------------------------------
# Function:    f_csr_hostlist_check_existence
#
# Description: This function checks if the file containing the list of iLOs
#              is existing and isn't empty.
#
# Parameters:  none
#
# Output:      none
#
# Return code: 0 - success (file exists)
#              1 - failure (file doesn't exist)
# ------------------------------------------------------------------------------

function f_csr_hostlist_check_existence ()
{
    local L_ILO_FQDN_LIST_FULL_PATH="${ILO_FQDN_LIST_FULL_PATH}";


    # check if the file is existing and not empty
    f_file_check_existence_non_empty "${L_ILO_FQDN_LIST_FULL_PATH}";
```

```
    return $?;
}


# ------------------------------------------------------------------------------
# Function:    f_csr_read_hostlist
#
# Description: This function reads the file containing the list of iLOs
#              line by line and adds it's contents to the global array
#              A_ILO_HOSTNAMES.
#
# Parameters:  none
#
# Output:      none
#
# Return code: 0 - success (file exists)
#              1 - failure (file doesn't exist)
# ------------------------------------------------------------------------------

function f_csr_read_hostlist ()
{
    local L_ILO_FQDN_LIST_FULL_PATH="${ILO_FQDN_LIST_FULL_PATH}";


    # read the file containing the list with FQDNs of iLOs into an array
    readarray A_ILO_HOSTNAMES < "$L_ILO_FQDN_LIST_FULL_PATH";

    return $?;
}


# ------------------------------------------------------------------------------
# Function:    f_csr_create_directory_structure
#
# Description: This function creates the necessary directory structure to store
# the CSRs in.
#
#
# Parameters:  none
#
# Output:      none
#
# Return code: none
# ------------------------------------------------------------------------------

function f_csr_create_directory_structure ()
{
    # walk through the array elements
    for L_FQDN in "${A_ILO_HOSTNAMES[@]}";
    do
        # strip non printable characters from the variable
        L_FQDN=$(tr -dc '[[:print:]]' <<< "${L_FQDN}");

        # create directory "script-var/CSRs/FQDN" if it doesn't already exist
        if [ ! -d "${CSRS_ROOT_DIR}/${L_FQDN}" ];
        then
            mkdir -p "${CSRS_ROOT_DIR}/${L_FQDN}" \
                && echo "INFO: created directory ${CSRS_ROOT_DIR}/${L_FQDN}" \
                || { echo "ERROR: couldn't create directory ${CSRS_ROOT_DIR}/${L_FQDN}"; exit 1; }
        fi
    done
}


# ------------------------------------------------------------------------------
# Function:    f_csr_fetch
#
# Description: This function sends a query to create a CSR to the FQDN which
#              gets passed to this function as the first argument.
#              The reply of the iLO gets stored in a global variable
```

```
#              "ILO_REPLY".
#
# Arguments:   1 - String - FQDN of an iLO
#
# Output:      none - This function sets the global variable "ILO_REPLY".
#
# Return code: none
# ------------------------------------------------------------------------------

function f_csr_fetch ()
{
    local L_FQDN="$1";
    local L_ILO_USER="${ILO_USER}";
    local L_ILO_PASS="${ILO_PASS}";
    local L_CSR_COUNTRY="${CSR_COUNTRY}";
    local L_CSR_STATE="${CSR_STATE}";
    local L_CSR_LOCALITY="${CSR_LOCALITY}";
    local L_CSR_ORGANIZATION="${CSR_ORGANIZATION}";
    local L_CSR_ORGANIZATIONAL_UNIT="${CSR_ORGANIZATIONAL_UNIT}";
    local L_CSR_STRING="";
    local L_ILO_REPLY="";


    # send the query to trigger the creation of a CSR to an iLO
    # via the tool "hpilo_cli"
    hpilo_cli -l ${L_ILO_USER} -p ${L_ILO_PASS} ${L_FQDN} \
                               certificate_signing_request \
                               country=${L_CSR_COUNTRY} \
                               state=${L_CSR_STATE} \
                               locality=${L_CSR_LOCALITY} \
                               organization="${L_CSR_ORGANIZATION}" \
                               organizational_unit=${L_CSR_ORGANIZATIONAL_UNIT} \
                               common_name=${L_FQDN} \
                               2> /dev/null;
}


# ------------------------------------------------------------------------------
# Function:    f_csr_check_for_todo_directory
#
# Description: This function checks if the todo directory exists.
#              If its not existing it will be created.
#
# Arguments:   none
#
# Output:      none
#
# Return code: 0 - success
#              1 - failure
# ------------------------------------------------------------------------------

function f_csr_check_for_todo_directory ()
{
    local L_FULL_PATH_TO_TODO_DIRECTORY="${DIR_TODO_LIST_DIR_PATH}";
    local L_STATUS="";


    # check if the todo directory exists and create it if it doesn't exist.
    if [ ! -d "${L_FULL_PATH_TO_TODO_DIRECTORY}" ];
    then
        mkdir -p "${L_FULL_PATH_TO_TODO_DIRECTORY}";
        L_STATUS=$?;
    fi
return $L_STATUS;
}

# ------------------------------------------------------------------------------
# Function:    f_csr_write_csr_to_file
#
# Description: This function creates
```

```
#
# Arguments:   1 - String - FQDN of an iLO
#
# Output:      none - This function sets the global variable "ILO_REPLY".
#
# Return code: 0 - success
#              1 - failure
# ------------------------------------------------------------------------------

function f_csr_write_csr_to_file ()
{
    local L_ILO_REPLY="$1";
    local L_FQDN="$2";
    local L_FULL_PATH_TO_CSR_FILE="${CSRS_ROOT_DIR}/${L_FQDN}/${L_FQDN}.csr";

    echo "${L_ILO_REPLY}" > "${L_FULL_PATH_TO_CSR_FILE}";

    # strip the first two lines from the file so that is only
    # the CSR left
    sed -i 1,1d "${L_FULL_PATH_TO_CSR_FILE}";

    if [ -e "${L_FULL_PATH_TO_CSR_FILE}" ];
    then
        echo "INFO: CSR written to ${L_FULL_PATH_TO_CSR_FILE}";
    else
        echo "ERROR: couldn't write CSR to ${L_FULL_PATH_TO_CSR_FILE}";
        exit 1;
    fi
return $?;
}

# ------------------------------------------------------------------------------
# Function:    f_csr_create_todo_files
#
# Description: This function creates the todo files
#
# Arguments:   none
#
# Output:      none - This function sets the global variable "ILO_REPLY".
#
# Return code: none
# ------------------------------------------------------------------------------

function f_csr_create_todo_files ()
{
    local L_FQDN="";
    local L_A_ILO_HOSTNAMES=("${A_ILO_HOSTNAMES[@]}");
    local L_FULL_PATH_TO_TODO_FILE="";

    for L_FQDN in "${A_ILO_HOSTNAMES[@]}";
    do
        # strip non printable characters from the variable
        L_FQDN=$(tr -dc '[[:print:]]' <<< "${L_FQDN}");

        # Build the full path to the todo file
        L_FULL_PATH_TO_TODO_FILE="${DIR_TODO_LIST_DIR_PATH}/${L_FQDN}";

        # if it doesn't already exist create the file
        if [ ! -e "${L_FULL_PATH_TO_TODO_FILE}" ];
        then
            touch "${L_FULL_PATH_TO_TODO_FILE}" \
                && echo "INFO: created todo file ${L_FULL_PATH_TO_TODO_FILE}" \
                || { echo "ERROR: couldn't create todo file ${L_FULL_PATH_TO_TODO_FILE}"; exit 1; }
        fi
    done
}


# ------------------------------------------------------------------------------
# Function:    f_csr_delete_todo_file
```

```
#
# Description: This function deletes a todo file
#
# Arguments:   1 - String - FQDN of an iLO
#
# Output:      none - This function sets the global variable "ILO_REPLY".
#
# Return code: 0 - success
#              1 - failure
# ------------------------------------------------------------------------------

function f_csr_delete_todo_file ()
{
    local L_FQDN="$1";
    local L_FULL_PATH_TO_TODO_FILE="${DIR_TODO_LIST_DIR_PATH}/${L_FQDN}";

    # strip non printable characters from the variable
    L_FQDN=$(tr -dc '[[:print:]]' <<< "${L_FQDN}");

    # delete a file (with the FQDN as the filename) as a todo list item.
    rm "${L_FULL_PATH_TO_TODO_FILE}" \
        && echo "INFO: deleted todo file ${L_FULL_PATH_TO_TODO_FILE}" \
        || { echo "ERROR: couldn't delete todo file ${L_FULL_PATH_TO_TODO_FILE}"; exit 1; }

return $?;
}


# ------------------------------------------------------------------------------
# Function:    f_csr_generate
#
# Description: This function connects to the configured Red Hat Satellite
#              server. There it fetches a list of Red Hat VMs running on
#              ProLiant hardware and then uses this list to generate a list of
#              FQDNs of iLOs which are built into those ProLiants.
#
# Parameters:  none
#
# Output:      none - The function sets the global variable ILO_FQDN_LIST
#
# Return code: none
# ------------------------------------------------------------------------------

function f_csr_generate ()
{
    local L_FQDN="";
    local L_ILO_COUNT="0";
    local L_ILO_FQDN_LIST_FULL_PATH="${ILO_FQDN_LIST_FULL_PATH}";
    local L_ILO_REPLY="";
    local L_FULL_PATH_TO_TODO_FILE="";
    local L_CSR_STATUS="0";
    local L_TODO_COUNTER="1";


    # check if the todo directory exists and create it if not.
    f_csr_check_for_todo_directory \
        && echo "INFO: todo directory exists." \
        || { echo "ERROR: todo directory doesn't exist or something went wrong trying to create it.";
exit 1; }


    # Check if the file containing the list of iLOs exists
    # and if it exists, read the hostlist into an array
    f_csr_hostlist_check_existence \
        && f_csr_read_hostlist \
        || { echo "ERROR: hostlist couldn't be read into the array"; exit 1; }


    # Count how many iLOs will be contacted to create CSRs and throw a message
    L_ILO_COUNT=$(wc -l < "${L_ILO_FQDN_LIST_FULL_PATH}");
```

```
    echo "INFO: ${L_ILO_COUNT} iLOs will be queried in the FIRST PASS.";


    # Build the TODO list
    f_csr_create_todo_files;


    # create the directory structure to store the CSRs in
    f_csr_create_directory_structure \
        && echo "INFO: CSR storage directory structure created." \
        || { echo "ERROR: directory structure couldn't be created"; exit 1; }


    # loop over the array with hostnames
    for L_FQDN in "${A_ILO_HOSTNAMES[@]}";
    do
        # reset L_CSR_STATUS
        L_CSR_STATUS="0";

        # strip non printable characters from the variable
        L_FQDN=$(tr -dc '[[:print:]]' <<< "${L_FQDN}");
        # tell the user that we are contacting an iLO
        echo "   ";
        echo "INFO: connecting to ${L_FQDN} (FIRST PASS: ${L_TODO_COUNTER} of ${L_ILO_COUNT})";
        # increment counter
        L_TODO_COUNTER=$((L_TODO_COUNTER+1))
        # connect to the iLO, send the request and collect what it will reply
        L_ILO_REPLY=$(f_csr_fetch "${L_FQDN}");

        # walk through what the iLO has replied, line by line, to find out if
        # the CSR has already been created.
        for L_LINE in "${L_ILO_REPLY}";
        do
            # we grep for the string "END CERTIFICATE REQUEST" because in case
            # the iLO hasn't created the CSR yet, the reply will not contain
            # that string.
            grep -q "END CERTIFICATE REQUEST" <<< "${L_LINE}";
            L_GREP_STATUS="$?";
            # if the grep above finds what it is searching for, we have
            # an already existing CSR on the iLO and we have to write it to
            # a file.
            if [ $L_GREP_STATUS -eq 0 ];
            then
                # Inform that we have a CSR
                echo "INFO: CSR already exists on ${L_FQDN}";
                # delete the todo file
                f_csr_delete_todo_file "${L_FQDN}";

                # set L_CSR_STATUS to 1
                L_CSR_STATUS="1";
            fi
        done

        # if we have found an already existing CSR, write it to a file
        # and inform if we haven't found a CSR yet
        if [ "$L_CSR_STATUS" -eq 1 ];
        then
            f_csr_write_csr_to_file "${L_ILO_REPLY}" "${L_FQDN}";
        else
            # Inform that we don't have a CSR yet
            echo "INFO: CSR doesn't exist on ${L_FQDN}";
        fi

    done

    # Build array with iLO FQDNs which have to be connected to again
    # to fetch the CSR (SECOND PASS)
    IFS=$'\n' A_STILL_TODO=($(ls ${DIR_TODO_LIST_DIR_PATH}/));
```

```
    # Count how many iLOs will be contacted again to collect CSRs
    # and throw a message
    L_ILO_COUNT=${#A_STILL_TODO[@]};
    echo "   ";
    echo "INFO: ${L_ILO_COUNT} iLOs will be queried in the SECOND PASS.";

    # reset L_TODO_COUNTER
    L_TODO_COUNTER="1";


    # loop over the array with iLOs from which we haven't collected a CSR yet
    for L_FQDN in "${A_STILL_TODO[@]}";
    do
        # reset L_CSR_STATUS
        L_CSR_STATUS="0";

        # strip non printable characters from the variable
        L_FQDN=$(tr -dc '[[:print:]]' <<< "${L_FQDN}");
        # tell the user that we are contacting an iLO
        echo "   ";
        echo "INFO: connecting again to ${L_FQDN} (SECOND PASS: ${L_TODO_COUNTER} of ${L_ILO_COUNT})";
        # increment counter
        L_TODO_COUNTER=$((L_TODO_COUNTER+1))
        # connect to the iLO, send the request and collect what it will reply
        L_ILO_REPLY=$(f_csr_fetch "${L_FQDN}");

        # walk through what the iLO has replied, line by line, to find out if
        # the CSR has already been created.
        for L_LINE in "${L_ILO_REPLY}";
        do
            # we grep for the string "END CERTIFICATE REQUEST" because in case
            # the iLO hasn't created the CSR yet, the reply will not contain
            # that string.
            grep -q "END CERTIFICATE REQUEST" <<< "${L_LINE}";
            L_GREP_STATUS="$?";
            # if the grep above finds what it is searching for, we have
            # an already existing CSR on the iLO and we have to write it to
            # a file.
            if [ $L_GREP_STATUS -eq 0 ];
            then
                # Inform that we have a CSR
                echo "INFO: CSR already exists on ${L_FQDN}";
                # delete the todo file
                f_csr_delete_todo_file "${L_FQDN}";

                # set L_CSR_STATUS to 1
                L_CSR_STATUS="1";
            fi
        done

        # if we have found an already existing CSR, write it to a file
        # and inform if we haven't found a CSR yet
        if [ "$L_CSR_STATUS" -eq 1 ];
        then
            f_csr_write_csr_to_file "${L_ILO_REPLY}" "${L_FQDN}";
        else
            # Inform that we don't have a CSR yet
            echo "WARNING: couldn't get a CSR for ${L_FQDN}";
        fi

    done


    # Build array with iLO FQDNs which we failed to create a CSR for
    IFS=$'\n' A_FAILED_ILOS=($(ls ${DIR_TODO_LIST_DIR_PATH}/));
    L_ILO_COUNT=${#A_FAILED_ILOS[@]};

    if [ $L_ILO_COUNT -ne 0 ];
    then
        # inform about failed hosts
```

```
        echo "   ";
        echo "WARNING:";
        echo "This script has failed to generate CSRs for the following FQDNs:";
        echo "   ";
        for L_FQDN in "${A_FAILED_ILOS[@]}";
        do
            echo "${L_FQDN}";
        done
    fi

exit 0;
}
```

```
#!/usr/bin/env bash

# ------------------------------------------------------------------------------
# Function:    f_file_check_existence_non_empty
#
# Description: This function checks if a file exists and is non empty.
#
# Arguments:   1 - full path to file
#
# Output:      none
#
# Return code: 0 - success (file exists and is not empty)
#              1 - failure (file doesn't exist or is empty)
# ------------------------------------------------------------------------------

function f_file_check_existence_non_empty ()
{
        local L_FULL_PATH_TO_FILE="$1";
        if [ -s "${L_FULL_PATH_TO_FILE}" ]
    then
        return 0;
    else
        return 1;
    fi
}


# ------------------------------------------------------------------------------
# Function:    f_formatted_date
#
# Description: This function generates a formatted string containing the
#              date and time of now (YYYYMMDD-HHMMSS)
#
# Arguments:   none
#
# Output:      String - formatted date
#
# Return code: 0 - success
#              1 - failure
# ------------------------------------------------------------------------------

function f_formatted_date ()
{
        local L_FORMATTED_DATE;

        L_FORMATTED_DATE=$(date +"%Y%m%d-%H%M%S");
        if [ $? -eq 0 ]
        then
            echo "${L_FORMATTED_DATE}";
                return 0;
        else
                return 1;
        fi
}
```

```
#!/usr/bin/env bash

# ------------------------------------------------------------------------------
# GLOBAL VARIABLES
# ------------------------------------------------------------------------------

ILO_FQDN_LIST="";     # Generated list with FQDNs of iLOs




# ------------------------------------------------------------------------------
# FUNCTIONS
# ------------------------------------------------------------------------------


# ------------------------------------------------------------------------------
# Function:    f_hostlist_get
#
# Description: This function connects to the configured Red Hat Satellite
#              server. There it fetches a list of Red Hat VMs running on
#              ProLiant hardware and then uses this list to generate a list of
#              FQDNs of iLOs which are built into those ProLiants.
#
# Parameters:  none
#
# Output:      none - The function sets the global variable ILO_FQDN_LIST
#
# Return code: none
# ------------------------------------------------------------------------------

function f_hostlist_get ()
{
    local L_SATELLITE_FQDN="${SATELLITE_FQDN}";
    local L_REMOTE_OUTPUT;
    local L_DEV_HOSTS_FQDN_LIST;
    local L_DE_HOSTS_FQDN_LIST;
    local L_FQDN_LIST;

    # connect via ssh and fetch a CSV which contains RHELs running on ProLiants at Location.
    # As an alternative this could be accomplished also with the REST-API which returns JSON:
    # $L_REMOTE_OUTPUT1=$(curl --insecure -u USERNAME:PASSWORD -H 'Accept:application/json'
https://demucvmr94.de.pri.domain.com/api/v2/hosts?search=+model++~+ProLiant+and++location++~+vmr);
    L_REMOTE_OUTPUT=$(ssh -q "${L_SATELLITE_FQDN}" "sudo hammer --csv --csv-separator ';' host list --
search 'location ~ vmr and model ~ ProLiant'");
    # Build a list with the FQDNs of the iLOs which belong to the RHEL systems
    # in "dev.de.pri.domain.com".
    # Drop first line, grep for ".dev.de.pri.domain.com", get only contents of second field,
    # replace ".dev.de.pri.domain.com" with "-adm.de.pri.domain.com".
    L_DEV_HOSTS_FQDN_LIST="$(echo "${L_REMOTE_OUTPUT}" | tail -n +2 | grep "\.dev\.de\.pri\.domain\.com"
| awk --field-separator=";" '{print $2}' | sed 's/\.dev\.de\.pri\.domain\.com/-
adm\.de\.pri\.domain\.com/')";

    # Build a list with the FQDNs of the iLOs which belong to the RHEL systems
    # in "de.pri.domain.com".
    # Drop the first line, drop all lines containing ".dev.de.pri.domain.com", get only contents of
    # the second field, replace ".de.pri.domain.com" with "-adm.de.pri.domain.com".
    L_DE_HOSTS_FQDN_LIST=$(echo "${L_REMOTE_OUTPUT}" | tail -n +2 | egrep -v
"\.dev\.de\.pri\.domain\.com" | awk --field-separator=";" '{print $2}' | sed 's/\.de\.pri\.domain\.com/-
adm\.de\.pri\.domain\.com/');

    # Set ILO_FQDN_LIST to the full list with all iLO FQDNs
    # by concatenating L_DEV_HOSTS_FQDN_LIST and L_DE_HOSTS_FQDN_LIST.
    ILO_FQDN_LIST=$(echo -e "${L_DEV_HOSTS_FQDN_LIST}\n${L_DE_HOSTS_FQDN_LIST}");
}

# ------------------------------------------------------------------------------
# Function:    f_hostlist_write
#
```

```
# Description: This function takes the contents of ILO_FQDN_LIST and writes it
#              to the path defined by ILO_FQDN_LIST_FULL_PATH.
#
# Parameters:  none
#
# Output:      none
#
# Return code: 0 - success
#              1 - failure
# ------------------------------------------------------------------------------

function f_hostlist_write ()
{
    local L_ILO_FQDN_LIST="${ILO_FQDN_LIST}";
    local L_ILO_FQDN_LIST_FULL_PATH="${ILO_FQDN_LIST_FULL_PATH}";
    # echo the contents of L_ILO_FQDN_LIST and redirect stdout to
    # the file which will contain the list
    echo -e "${L_ILO_FQDN_LIST}" > "${L_ILO_FQDN_LIST_FULL_PATH}";

    return $?;
}


# ------------------------------------------------------------------------------
# Function:    f_hostlist_check_existence
#
# Description: This function checks if the file containing the list of iLOs
#              is existing and isn't empty.
#
# Parameters:  none
#
# Output:      none
#
# Return code: 0 - success (file exists)
#              1 - failure (file doesn't exist)
# ------------------------------------------------------------------------------

function f_hostlist_check_existence ()
{
    local L_ILO_FQDN_LIST_FULL_PATH="${ILO_FQDN_LIST_FULL_PATH}";

    # check if the file is existing and not empty
    f_file_check_existence_non_empty "${L_ILO_FQDN_LIST_FULL_PATH}";

    return $?;
}



# ------------------------------------------------------------------------------
# Function:    f_hostlist_backup
#
# Description: This function creates a backup of the file which contains the
#              list of iLO FQDNs.
#
# Parameters:  none
#
# Output:      none
#
# Return code: 0 - success (file exists)
#              1 - failure (file doesn't exist)
# ------------------------------------------------------------------------------

function f_hostlist_backup ()
{
        local L_ILO_FQDN_LIST_FULL_PATH="${ILO_FQDN_LIST_FULL_PATH}";
        local L_FORMATTED_DATE=$(f_formatted_date);
        # copy the list with iLO-FQDNs to <filename>-YYYYMMDD-HHMMSS
        cp -a "${L_ILO_FQDN_LIST_FULL_PATH}" "${L_ILO_FQDN_LIST_FULL_PATH}-${L_FORMATTED_DATE}";
```

```
        return $?;
}


# ------------------------------------------------------------------------------
# Function:    f_hostlist_cleanup_backups
#
# Description: This function cleans up the backups by deleting files which are
#              older than 186 days.
#
# Parameters:  none
#
# Output:      none
#
# Return code: 0 - success
#              1 - failure
# ------------------------------------------------------------------------------

function f_hostlist_cleanup_backups ()
{
        local L_DIR_ILO_FQDN_LISTS_PATH="${DIR_ILO_FQDN_LISTS_PATH}";
        find "${L_DIR_ILO_FQDN_LISTS_PATH}" -mtime "${HOSTLISTS_RETENTION_TIME}" -type f -delete;

        return $?;
}


# ------------------------------------------------------------------------------
# Function:    f_hostlist_generate
#
# Description: This function
#
# Parameters:  none
#
# Output:      The function generates the file ilo_hostlist.lst in the scripts
#              var-directory.
#
# Return code: 0 - success
#              1 - failure
# ------------------------------------------------------------------------------

function f_hostlist_generate ()
{
    # fetch a list containing the FQDNs of our iLOs
    f_hostlist_get;
    # check if the file, to which the list should be written, already exists
    f_hostlist_check_existence;

    # back it up if the file already exists
    if [ $? -eq 0 ]
    then
        f_hostlist_backup;
    fi

    # write the list out to a file
    f_hostlist_write;

    # cleanup old hostlists
    f_hostlist_cleanup_backups;
}
```

```
#!/usr/bin/env bash

# ------------------------------------------------------------------------
# This file includes all necessary files from the script's lib-directory
# ------------------------------------------------------------------------

# List of files which should get included
```

```
LIB_SCRIPTS="zertifix_arguments_parser.sh
zertifix_generic_functions.sh
zertifix_hostlist_functions.sh
zertifix_csr_functions.sh
zertifix_controller.sh";
# Include all files listed above in a loop
for LIB_INCLUDE in ${LIB_SCRIPTS}
do
    source "${SCRIPT_LIB}/${LIB_INCLUDE}";
done;
```

```
-----BEGIN CERTIFICATE REQUEST-----
MIIDBTCCAe0CAQAwgYUxJTAjBgNVBAMMHGRlbXVjMWRyMDEtYWRtLmRlLnByaS5v
Mi5jb20 POSSIBLY MAYBE OR THE OTHER WAY AROUND uaWNhIEdlcahbnkas
R21iSDEPMA0Ga1UEBwwGTXVuaWNoMQ8wDQYDVQQIDAZCYXllcm4xCzAJBgNVBAYT
AkRFMIIBIjAN FAKE DATA CgKCAQEA3kPs03 POSSIBLY sdf8ljjZEEN3r+jVf
HNTBN3qQz3NBN FITZE FITZE FATZE FITZE FITZE FATZ 0XZ1psFwt4ATncd
i5ei2+Q9v3WrTHM524jcTbp1UAfS3vG5ptsD7Li9PMovKUxJtswAs0PRmVpRc5pO
BB7dnsi+Uw8cKAUYXT7lA+3OcfPLPpKMkFpTNMU2gyJ4yBdLo3XTLOZSEfNTOzKc
NKv4mY5ze3hWp THISISJUSTATEST G3mtqDN80hqcV0Guxoa9AOoL LAlaLAlaA
E8IDFX7Y5WAZZnQXv0c+pHrW1uT2Xkb+qr0b2Qz4bICrrd7LdDp6n2uMMh8OCyKi
v7n+hdf8psöfgj09870ilkjö-älöd0LKJÖLßkjäG=öÖööäöäÖdjsg999ZW11YzFk
cjAxLWFkbS5kZS5wcmkubzIuY29tMA0GCSqGSIb3DQEBCwUAA4IBAQBDEYy7QzNR
WSmmvgRG1hws1vzPP9amTiGaW38H9NMbyuQ0AXkn7R/eCGH2awsjidjA+M5V86/c
m6qIyEnMYcfSoFN5 ARFF DEAD:BEEF::   safdaK2aPzX8d1wmTcmsrUfEc/tW
AfA09pD/n+P7bSAVJEt67ekWgjkPR0cabGTdxBQcjaDNsKOAQae/sW4r642hN/gY
amZuPgSWI/dW+kHlg94Pr0npww9sK2X8FrBENvIuYX2yHyO75nnjmO2MtgZU51hL
bKjGBh1zxyGrfsf8iösg90=FA=Fioafj909sfolllksfß0sfllkk9987lN7+Ux7z
KpsHHLR05B/L
-----END CERTIFICATE REQUEST-----
```