OPEN DIGITAL
CONSULTING CO

Platform functional documentation

HOX project

Stuart J Mackintosh

Wednesday 30 July 2025

Contents

1 How to Write Documentation for Bash Functions

This guide explains how to document your Bash functions in a way that is clear for other developers and compatible with automated documentation extraction tools. Following these best practices will ensure your scripts are easy to understand and maintain.

1.1 Best Practice: Function Documentation Format

Place a **block of comment lines immediately above each function definition**. This block should provide all the key details about the function.

Template:

```
# function_name: Brief description of what the function does.

# Globals:
# VAR_NAME - Description of any global variables used or modified.

# Arguments:
# \$1 - Description of the first argument.
# \$2 - Description of the second argument.

# Outputs:
# Description of outputs (e.g., "Writes to STDOUT", "Creates a file").

# Returns:
# 0 on success, non-zero on error.

function_name() {
# Function implementation...
}
```

Example:

```
# greet_user: Prints a personalized greeting.

# Globals:
```

```
# PREFIX - Greeting prefix (default: "Hello")

# Arguments:
# \$1 - Name to greet (string)

# Outputs:
# Writes greeting to STDOUT.

# Returns:
# 0 on success, non-zero on error.

greet_user() {
local name="$1"
  echo "${PREFIX:-Hello}, \$name!"
}
```

1.2 Key Points

- **Place documentation immediately before the function definition**—no blank lines between the comment block and the function.
- **Use # for each line** of the documentation block[^8_2].
- **Describe all arguments** using `$1`, `$2`, etc., and indicate their purpose and type.
- **List any global variables** the function uses or modifies.
- **Explain outputs** (what is printed, written, or returned).
- **State return values** and their meaning.
- **Keep descriptions concise but informative**.

1.3 Additional Best Practices

- **Use consistent labels**: `Globals`, `Arguments`, `Outputs`, `Returns`.
- **Name function arguments with `local` variables** inside the function for clarity[^8_3].
- **Update documentation** whenever the function changes.
- **Avoid redundant comments**; focus on information not obvious from the code itself.
- **Use consistent formatting and indentation** for readability.
- **Consider a naming convention** (e.g., underscores, prefixes) to avoid clashes[^8_3].
- **Group all functions near the top** of your script, before the main code[^8_4].

1.4 Why This Matters

- **Clarity**: Makes your code easier to understand for others and your future self.
- **Automation**: Allows tools to extract and present documentation automatically.
- **Consistency**: Following a standard style improves maintainability and reduces confusion[^8_4].

By following this structure, your function documentation will be easy for both humans and automated tools to read and extract.

2 Function documentation

2.1 bootstrap_initialise_distro

Contained in `lib/functions.d/configure-distro.sh`

2.1.1 1. Function Overview

The `bootstrap_initialise_distro` function is primarily designed for system provisioning tasks in a Unix-based operating system (OS). This function initializes the OS distribution by injecting specific bootstrap instructions. It accepts a machine address (MAC) as a parameter and uses it to generate a shell script with bootstrap instructions.

2.1.2 2. Technical Description

- **Name:** `bootstrap_initialise_distro`
- **Description:** This function creates a bash script with bootstrap initialization instructions for a Unix-like OS. It uses a `cat` command to output a standard bash script header followed by specific bootstrapping instructions. A MAC address may be provided to customize the bootstrap instructions, although the current function version does not seem to be using this parameter.
- **Globals:** None
- **Arguments:**
  - `$1: mac` The MAC address of the target machine
- **Outputs:** A bash script with bootstrap instructions
- **Returns:** No value returned
- **Example usage:** `bootstrap_initialise_distro "00:0a:95:9d:68:16"`

2.1.3 3. Quality and Security Recommendations

- The function currently doesn't use the passed MAC address in any way. Depending on the use case, consider incorporating it into the bootstrap script for device-specific configuration.
- Take care to sanitize any user-provided values such as MAC addresses to prevent injection attacks.

- The output of the `cat` command isn't currently captured or directed in any way. Consider redirecting it to a file or capturing it in a variable for later use.
- Document the expected bootstrap procedures in the function's comments, so that it's clear to users and maintainers what this script is expected to do.
- Finally, ensure that the function's usage is limited to users with appropriate permissions. The creation of a bootstrapping script should typically be restricted to system administrators or roles with equivalent privileges.

2.2 build_yum_repo

Contained in `lib/functions.d/repo-functions.sh`

2.2.1 Function overview

The `build_yum_repo` function is a Bash function designed to create or update a YUM repository. This function validates the provided input, checks for the presence of the `createrepo_c` command, checks for changes in RPM files in the given directory, updates the repository if changes are detected or no previous state exists, and stores the current state.

2.2.2 Technical description

- **name**: build_yum_repo
- **description**: The function is used within a Bash script to create or update a YUM repository.
- **globals**: None
- **arguments**:
  - $1: repo_path - The path string to the directory containing RPM files to be checked and potentially updated in the repository.
- **outputs**: All logs are outputs and are either an error message regarding a missing variable or directory, a status message about the function's progress, or a success message stating the repository's successfully built.
- **returns**: The function can return 3 potential values. If there is an issue with the provided path variable or the directory does not exist, the function will return 1. If the `createrepo_c` command is missing, the function will return 2. If the function successfully creates or updates the repository, or if there are no changes needed, it will return 0.
- **example usage**:

```
build_yum_repo "${HPS_PACKAGES_DIR}/${DIST_STRING}/Repo"
```

2.2.3 Quality and security recommendations

1. Consider adding more comments within the function to increase readability.
2. Enhance error handling. For instance, handle the scenario where the script does not have write access to the check-sum file or the repository path.
3. Validate the `createrepo_c` command's successful installation by checking its return value rather than the presence of the command.
4. Verify the RPM files' integrity in the repository path, if not checked elsewhere.
5. Consider using more descriptive names for local variables. It would increase the readability and maintainability of the code.
6. Be sure to keep all software up-to-date, including the `createrepo_c` package, to ensure you have the latest security patches.

2.3 `cgi_header_plain`

Contained in `lib/functions.d/cgi-functions.sh`

2.3.1 Function Overview

This is the `cgi_header_plain` function included in the Bash scripts. This small function generates a common gateway interface (CGI) header in plain text. This header is typically used to inform the HTTP server about the type of content it is receiving or sending. In this case, it informs the server that the content it is about to send or receive is in plain text format using the `Content-Type` HTTP header field.

2.3.2 Technical Description

Name: `cgi_header_plain`

Description: The `cgi_header_plain` function prints out a plain text CGI header. It's typically used to inform the HTTP server that the content is in plain text.

Globals: None

Arguments: None

Outputs: Prints to stdout:

```
Content-Type: text/plain
```

Returns: None

Example Usage:

```
cgi_header_plain
```

2.3.3 Quality and Security Recommendations

1. Security: Be cautious about directly echoing any user-provided input within your `cgi_header_plain` function as this could potentially open up cross-site scripting (XSS) problems.
2. Quality: Ensure there is enough error handling/logging to deal with any failures that may occur while executing the function.
3. Quality: Document the function usage and any side effects to prevent misuse by other developers.
4. Security: Always consider the potential risk of injection attacks and apply necessary preventative measures to mitigate them.
5. Quality: Provide a simple, clear, and concise function definition for better readability and maintainability.
6. Security: Make sure the function only has the minimum required permissions necessary to perform its task, following the principle of least privilege.

2.4 `cgi_log`

Contained in `lib/functions.d/cgi-functions.sh`

2.4.1 Function Overview

The `cgi_log` function is used to log messages into a CGI log file. It takes a single argument, a message from the user, and appends this message to the `/var/log/ipxe/cgi.log` file along with the current timestamp.

2.4.2 Technical Description

- **Name:** `cgi_log`
- **Description:** The function accepts a user-provided logging message and appends it in a pre specified log file with the current timestamp.
- **Globals:** None.
- **Arguments:** [`$1`: The log message provided by the user]
- **Outputs:** Appends a log message to the `/var/log/ipxe/cgi.log` file.
- **Returns:** No explicit return value.
- **Example Usage:** `cgi_log "This is a sample log message"`

2.4.3 Quality and Security Recommendations

- Sanitize the input log message in order to prevent any form of code injection or any other malicious activity.

- Provide error messages if the user does not provide a log message or if the log file is not accessible or writable.
- Be sure to handle errors and exceptions, such as a full disk or inaccessible file.
- Log messages should be meaningful and contain relevant information for easier debugging, if need be.
- From a security perspective, it is important to restrict read/write permissions for log files to prevent unauthorized access.
- Consider including a check for log file size to prevent it from consuming excessive disk space.
- Timestamp should be in a common format which should be timezone aware.

2.5 `cgi_param`

Contained in `lib/functions.d/cgi-functions.sh`

2.5.1 Function overview

The `cgi_param` function is a tool for managing CGI parameters in a Bash script environment. It reads from a query string and saves the validly formatted key-value pairs into a global associative array. The function also supports methods for retrieving the value of a specified key (command “get”), checking if a specified key is present (command “exists”), and validating if a key’s value equals to a given value (command “equals”). An invalid command results in an error message and a returned value of 2.

2.5.2 Technical description

- **name**: `cgi_param`
- **description**: The function processes CGI parameters passed as a query string. It supports handling command and key-value pairs, reading from query string only once, dispatching commands (get, exists, equals), and error handling when an invalid command is put in.
- **globals**: [`__CGI_PARAMS_PARSED`: A flag indicating whether the query string has been parsed, `CGI_PARAMS`: An associative array holding key-value pairs from the parsed query string]
- **arguments**: [`$1`: The command, `$2`: The key, `$3`: An optional value used for the “equals” command]
- **outputs**: For the command “get”, if the key exists, it prints the corresponding value. For an invalid command, it prints an error message.
- **returns**: No specific return value. Overall result mainly depends on the success of the commands run.
- **example usage**: `cgi_param "get" "example"`

2.5.3 Quality and security recommendations

- Apply more rigorous input validation and error handling where required to ensure the robustness and reliability of the script.
- Avoid using environment variables (QUERY_STRING) directly, as they can be very unpredictable.
- Be aware of the risk of command injection as the script has the potential to be exploited.
- Restrict the regex patter that validates the decoded_key to include only what's necessary in order to avoid potential security vulnerabilities.
- Use a more descriptive name for the function for better readability and maintainability.
- Always initialize your variables.
- Validate key before using it, as it might not be in a valid format.

2.6 cgi_success

Contained in lib/functions.d/cgi-functions.sh

2.6.1 Function overview

The cgi_success() is a bash function used within the scope of a CGI (Common Gateway Interface) based application. This function is intended to produce a plain text response from the web server back to the client. Initially, it calls another function, cgi_header_plain, which is used to output the proper HTTP headers to make the response be treated as plain text. Afterward, the function outputs the message passed to it as an argument.

2.6.2 Technical description

- **Name:** cgi_success
- **Description:** This bash function emits plain text response for CGI scripts. It begins by calling cgi_header_plain to set necessary HTTP headers for the response to be interpreted as plain text. Following that, it echoes the argument passed to it.
- **Globals:** None.
- **Arguments:** [$1: The message or data to be displayed as plain text in the HTTP response]
- **Outputs:** The function outputs the HTTP headers necessary for a plain text response as well as the data or message passed as an argument.
- **Returns:** None.
- **Example Usage:**

```
cgi_success "Operation completed successfully"
```

This will output the plain text “Operation completed successfully” along with the appropriate HTTP headers dictated by the `cgi_header_plain` function.

2.6.3 Quality and Security Recommendations

- Validate and sanitize the input argument to prevent the Cross-Site Scripting (XSS) vulnerability since we are dealing with an output that is directly rendered on a browser.
- Document the `cgi_header_plain` function that this function calls to better understand what header fields are set, and ensure they are appropriately configured.
- Include error-handling mechanisms to ensure that the function behaves as expected even in the event of errors or exceptions.
- It’s good practice to always use double quotes around the variable (`"$1"`) to prevent word splitting and pathname expansion.
- Check if the argument is provided to the function before echoing it. If no argument is provided the function should handle it gracefully.
- Use a more informative function name that better describes the intended action(s).

2.7 `check_and_download_latest_rocky`

Contained in `lib/functions.d/iso-functions.sh`

2.7.1 Function Overview

The function `check_and_download_latest_rocky` is designed to check for the latest version of Rocky Linux available for the x86_64 architecture and download the minimal ISO file if not already present on the system. The function uses cURL to download the ISO file, if not found locally. It also log the latest version number for debugging purposes and creates necessary directories for storing ISO files.

2.7.2 Technical Description

- **Name:** check_and_download_latest_rocky
- **Description:** Checks for the latest version of Rocky Linux available and downloads the ISO file if not present in the local system. The downloaded ISO is minimal for `x86_64` architecture.
- **Globals:**
  - HPS_DISTROS_DIR: The directory in which the ISO file will be stored.
- **Arguments:** No arguments required.
- **Outputs:** Downloads the ISO file. Prints the status of ISO file (whether downloading or already present).
- **Returns:** Does not return anything but exits with status 1 if the latest version is not detected.

- **Example Usage:**

```
check_and_download_latest_rocky
```

2.7.3 Quality and Security Recommendations

- Always validate the URL before using cURL for downloads.
- Implement error handling for failed cURL downloads and directory creations.
- Check if the global variable `HPS_DISTROS_DIR` is set before the function is called.
- Consider adding an argument to specify the architecture or ISO type gaining more flexibility.
- For security, consider verifying the checksum of the downloaded ISO to ensure it is not tampered with.

2.8 `check_latest_version`

Contained in `lib/functions.d/iso-functions.sh`

2.8.1 Function overview

The function `check_latest_version` is used to check for the latest version of an operating system. The function takes three parameters: cpu (processor type), mfr (manufacturer), and osname (operating system name). It fetches the HTML from the base URL of the OS, parses the page for versions, and echoes the latest version found. For instance, it can access the base URL of the operating system 'Rocky Linux' and echo the latest version if such exists. If an error occurs during the fetch or if no versions are found, an error message is displayed and the function returns 1. In case the OS provided is unknown, the function also echoes an error message and returns 1.

2.8.2 Technical description

- name: `check_latest_version()`
- description: The function checks for the latest version of an operating system.
- globals: None
- arguments:
  - `$1:` `cpu` - the type of cpu
  - `$2:` `mfr` - the manufacturer
  - `$3:` `osname` - the name of the operating system
- outputs: Echoes the status of version checking, any potential error messages, and the latest version number if found.

- returns: Returns 1 when an error occurs, or 0 on successful finding of the latest version.
- example usage:

```
check_latest_version "x86_64" "Intel" "rockylinux"
```

2.8.3 Quality and security Recommendations

- To improve the security of this function, it's recommended to include additional mechanisms for validating the security certificates of the pages you fetch through `curl`.
- As a quality measure, it might be beneficial to add support for other operating systems or at least output a more specific error message when an unavailable OS is supplied.
- Considering the potential changes over time on the HTML structure of the page that this function scrapes, it'd be recommended to maintain and adapt the parsing method according to these changes accordingly.
- Additional error checks should be added to ensure that the function parameters are not empty before the function attempts to operate with them.
- On a practical note, it would be optimal to avoid hard-coding the base URL for each OS and instead, maybe, fetch it from a maintained list or database.

2.9 `cluster_config`

Contained in `lib/functions.d/cluster-functions.sh`

2.9.1 Function Overview

The `cluster_config` function is used as a utility to interact with an active cluster configuration file. It has three operations: get, set, and exists. The `get` operation retrieves the value for a specified key from the cluster file. The `set` operation adds or updates a key-value pair in the cluster file. Lastly, the `exists` operation checks if a particular key exists in the cluster file.

2.9.2 Technical Description

- Name: `cluster_config`
- Description: A utility function to get, set, or check if a key-value pair exists in an active cluster configuration file.
- Globals: None.
- Arguments:
  - `$1`: This is the operation to be performed: get, set, or exists.
  - `$2`: This is the key that is being manipulated or queried.

  - `$3`: This is the value that is used when the 'set' operation is selected. It is an optional parameter with a default empty string.
- Outputs:
  - Value of a defined key if the 'get' operation is selected.
  - Confirmation of an update or addition if the 'set' operation is selected.
  - Existence logics if the 'exists' operation is selected.
  - An error message if an invalid operation is chosen or if there's no active cluster config.
- Returns:
  - `1` if no active cluster configuration is found.
  - `2` if an invalid operation is chosen.
- Example usage:

```
cluster_config get nodeSize
cluster_config set masterNode 'Master-1'
cluster_config exists nodeSize
```

2.9.3 Quality and Security Recommendations

1. Sanitizing inputs: Since the values of the variables could come from untrusted sources, it would be sensible to sanitize these values before they are used.
2. Error checking: Beyond the three defined operations (get, set and exists), an error gets thrown but with a generic message. It would be good to have a separate error message for absent cluster config and invalid operation selected.
3. Logging: More detailed logging might be useful for debugging purposes.
4. Commenting: More detailed commenting throughout the function will make the function easier to maintain.
5. Function validation: Check if `cluster_config` function actually exists before calling it.

2.10 `cluster_has_installed_sch`

Contained in `lib/functions.d/cluster-functions.sh`

2.10.1 Function overview

The Bash function `cluster_has_installed_sch()` checks whether a cluster has an installed "SCH" type. The function reads through cluster configuration files with `.conf` extension in a given directory and checks each of them if it has a line corresponding to type "SCH" and state "INSTALLED". If there is indeed an installed "SCH" type, it returns 0 (true - SCH is installed), otherwise, it returns 1 (false - SCH is not installed).

2.10.2 Technical description

- **name**: `cluster_has_installed_sch`
- **description**: This function loops through each .conf file in a given directory. Each configuration file is scanned for the type "SCH" and state "INSTALLED". If a configuration with this specific type and state is found, the function returns 0, otherwise, it returns 1.
- **globals**: [`HPS_HOST_CONFIG_DIR`: The directory where the configuration files (.conf extension) are stored]
- **arguments**: none
- **outputs**: No explicit output is produced, but the function changes the exit status to indicate the presence of an installed "SCH" type.
- **returns**: Returns 0 if an installed "SCH" type is found, 1 if not.
- **example usage**:

```
if cluster_has_installed_sch; then
  echo "SCH type is installed"
else
  echo "SCH type is not installed"
fi
```

2.10.3 Quality and security recommendations

1. Make sure to set the global variable `HPS_HOST_CONFIG_DIR` prior to calling the function, otherwise it won't find the files to process.
2. For better security, use absolute paths when setting `HPS_HOST_CONFIG_DIR` to avoid reliance on relative paths which could be exploited.
3. Validate the content of the configuration files to avoid injections.
4. In case of read failures or other I/O problems, those should be handled graciously.
5. Adding comments to the code would improve its maintainability by making it easier for others to understand.
6. Consider adding more error checks e.g., if directory or files do not exist.

2.11 `configure_dnsmasq`

Contained in `lib/functions.d/configure_dnsmasq.sh`

2.11.1 Function overview

The `configure_dnsmasq` function is responsible for setting up and configuring dnsmasq on a DHCP IP. This function first checks if the DHCP IP exists in the active cluster, then generates a configuration file for dnsmasq and copies certain necessary files to the TFTP directory. If the DHCP IP does not exist, the function exits and outputs an error message.

2.11.2 Technical description

- **Name**: `configure_dnsmasq`
- **Description**: This function sets up dnsmasq configuration on a DHCP IP. It checks if the IP exists, generates a dnsmasq configuration file, and copies necessary iPXE files to the TFTP directory.
- **Globals**:
  - `DHCP_IP`: The IP address used for DHCP.
  - `HPS_SERVICE_CONFIG_DIR`: Directory for service configuration files.
  - `HPS_TFTP_DIR`: Directory for TFTP.
- **Arguments**:
  - None
- **Outputs**: Messages indicating the progress and result of the configuration.
- **Returns**: No return value, but it does exit the script if the DHCP IP is not present.
- **Example usage**: `configure_dnsmasq`

2.11.3 Quality and security recommendations

- It is recommended to include more error handling for cases where required files are not successfully copied to the TFTP directory.
- Consider sanitizing or validating the `DHCP_IP` and the `NETWORK_CIDR` to prevent potential security issues.
- The references to the external file paths should be validated that they exist and are readable before attempting to use them.
- Some global variables' values are directly inserted into configuration files, it's recommended to verify and sanitize these values to make sure they do not contain any malicious or unexpected inputs.
- It would be ideal to implement good practices for creating secure temporary files. Examples are: using `mktemp` for creating temporary files and explicitly setting proper file permissions.

2.12 `configure_ipxe`

Contained in `lib/functions.d/configure_ipxe.sh`

2.12.1 Function overview

The `configure_ipxe` function is used to create a boot configuration file for iPXE, an open-source network boot firmware. The configuration file is written to a specific directory, which is designated by the `HPS_MENU_CONFIG_DIR` global variable. The function also uses the `DHCP_IP` global variable to specify the server IP address within the configuration file. It creates a variety of boot options and procedures within the configuration file, enabling the iPXE server to provide network boot services.

2.12.2 Technical description

- **Name:** configure_ipxe
- **Description:** The function creates a configuration file for a network boot firmware, iPXE. It sets up the file to respond to different conditions and boot options based on the client IP and MAC addresses.
- **Globals:** [HPS_MENU_CONFIG_DIR: A global variable indicating the directory where the configuration file should be written. DHCP_IP: A global variable containing the IP address of the DHCP server.]
- **Arguments:** [None]
- **Outputs:** The function writes to a boot.ipxe configuration file in the specified directory.
- **Returns:** No value returned. The success of the function can be determined by the existence and correctness of the output file.
- **Example usage:** `configure_ipxe`

2.12.3 Quality and security recommendations

- **Validate user input:** Make sure that all user input, such as the `HPS_MENU_CONFIG_DIR` and `DHCP_IP` global variables, are validated before use.
- **Use secure file permissions:** Ensure that the resulting configuration file has appropriate file permissions to prevent unauthorized access or modification.
- **Error Handling:** Add error handling to catch and handle potential errors or exceptions. For example, ensure that the directory exists before trying to write the file.
- **Code comments:** Improve code readability and maintainability by adding comments that describe the purpose and function of the code blocks.
- **Encryption or obfuscation:** If sensitive data is transmitted, consider using encryption or obfuscation to protect this data.
- **Regularly update software:** Regularly update iPXE and related software packages to ensure they have the latest security patches.

2.13 `configure_kickstart`

Contained in `lib/functions.d/configure_kickstart.sh`

2.13.1 Function Overview

The bash function `configure_kickstart` defined above is used for generating a Kickstart configuration file for a given cluster in a high performance computing environment. The function takes one required parameter, the name of the cluster, constructs the path to the Kickstart file for the cluster and generates the Kickstart configuration file

at the constructed path. If the required argument is not provided, the function exits and displays an error message.

2.13.2 Technical Description

Name

`configure_kickstart`

Description

Generates a Kickstart configuration file for a specified cluster.

Globals - `CLUSTER_NAME`: The name of the cluster for which the Kickstart file is to be generated. - `KICKSTART_PATH`: The path where the Kickstart file for the cluster is to be generated.

Arguments - `$1`: Represents the name of the cluster. Required for the execution of the function.

Outputs

Prints messages about the process to stdout, including an error if the cluster name is omitted and a confirmation message when the Kickstart file is successfully generated.

Returns

Returns 1 and exits if the required cluster name argument is missing.

Example Usage

```
configure_kickstart my_cluster
```

2.13.3 Quality and Security Recommendations

1. Improve error messages: The error message upon missing cluster name could be more descriptive and provide a suggestion to the user to supply the required parameter.
2. Input validation: The function may include validation of the cluster name to ensure it does not contain special characters that could lead to unexpected issues.
3. Sensitive information: The function embeds a sysadmin user with a default password directly in the Kickstart. Consider fetching such credentials from secure storage or prompting the user for them during runtime to minimize the risk of the credentials being compromised.
4. Fault tolerance: Error handling could be introduced to manage scenarios where file creation fails due to insufficient permissions or disk space issues.
5. Potentially provide options for the network configuration, partition configuration, package selection, etc., making the function more flexible and customizable for different use cases.

2.14 configure_nginx

Contained in lib/functions.d/configure_nginx.sh

2.14.1 Function overview

The function configure_nginx is a Bash function designed to set up configuration settings for the Nginx web server for a specific cluster environment. The function's main functionality is to direct standard error to null and source the active cluster filename. It also defines file paths and presets the Nginx configuration file nginx.conf.

2.14.2 Technical description

- **Name**: configure_nginx
- **Description**: This function initializes Nginx configuration for the active cluster environment. It sources the filename of the active cluster, which it retrieves via the get_active_cluster_filename function. It then creates (or overwrites) the nginx.conf file using a heredoc, setting certain preset configurations.
- **Globals**:
  - HPS_SERVICE_CONFIG_DIR: The directory containing service configuration files.
- **Arguments**: None
- **Outputs**: A nginx.conf file in the directory specified by HPS_SERVICE_CONFIG_DIR, containing basic Nginx configuration parameters.
- **Returns**: No return value.
- **Example Usage**: bash configure_nginx

2.14.3 Quality and security recommendations

- It's important to remember to handle errors appropriately and not just direct them to null. You may want to add error handling for when the get_active_cluster_filename function fails to retrieve a filename.
- The function currently assumes that the global variable HPS_SERVICE_CONFIG_DIR is correctly set. It would be beneficial to validate this assumption before using the variable.
- Hard-coding configuration settings for Nginx within the function may not be ideal for differing environments. Consider parameterizing these settings or retrieving them from an outside source to increase flexibility.
- Redirecting standard error to null can be risky as it may overlook potential problems, consider logging errors instead.
- It is recommended to consistently use double quotes around variable references to prevent word splitting and filename expansion.
- Make sure that only authorized users have write access to the nginx.conf file, as unauthorized access may lead to security vulnerabilities.

- For overall secure coding practices, refer to the OWASP Secure Coding Practices.

2.15 configure_supervisor_core

Contained in `lib/functions.d/configure-supervisor.sh`

2.15.1 Function overview

The function `configure_supervisor_core` generates a supervisord configuration file at the location specified by the `SUPERVISORD_CONF` environment variable. This function also creates the directory `/var/log/supervisor` if it doesn't already exist. The configuration file includes settings for supervisord, supervisorctl, rpcinterface, and unix_http_server. Message output is provided to track the function's progress.

2.15.2 Technical description

- **Name**: `configure_supervisor_core`
- **Description**: The function generates a supervisord configuration file with mandatory fields.
- **Globals**: [`HPS_SERVICE_CONFIG_DIR`: The directory where the supervisord configuration file is to be stored.]
- **Arguments**: None.
- **Outputs**: Prints messages on stdout about the function's progress and completion.
- **Returns**: None.
- **Example usage**: The function is used as `configure_supervisor_core`

2.15.3 Quality and security recommendations

1. Use more descriptive names for variables, and include comments explaining what each variable is used for.
2. Should configure error handling so if the supervisor service can't be created for any reason, it returns a meaningful error message.
3. The "admin" username is hardcoded, it is recommended to store usernames in environment variables or some sort of secure and encrypted configuration that can be loaded at runtime.
4. The password is currently hardcoded as "ignored-but-needed", this should be replaced with a more secure method, such as a randomly generated password or one provided securely at runtime. Alternatively, it could be removed if it isn't needed.
5. Consider setting the supervisor to run as a non-root user for increased security.
6. Use explicit file permissions when creating directory to prevent unauthorized access.

2.16 configure_supervisor_services

Contained in lib/functions.d/configure-supervisor.sh

2.16.1 Function overview

The configure_supervisor_services is a Bash function that is responsible for creating a Supervisor services configuration file known as supervisord.conf. This function first calls configure_supervisor_core, then appends configurations for three programs: dnsmasq, nginx, and fcgiwrap to supervisord.conf.

2.16.2 Technical description

- **name**: configure_supervisor_services
- **description**: Generates a Supervisor services configuration file and appends the setups needed to run three services: dnsmasq, nginx, and fcgiwrap.
- **globals**: [HPS_SERVICE_CONFIG_DIR: A string storing the path to the directory where the Supervisor config files are kept.]
- **arguments**: None
- **outputs**: A supervisord.conf file in the directory specified by HPS_SERVICE_CONFIG_DIR, containing configurations for three services.
- **returns**: None
- **example usage**: configure_supervisor_services

2.16.3 Quality and security recommendations

1. To avoid any misconfiguration or to handle any non-existent directory paths, the function should be improved by adding error handling logic when updating the supervisord.conf file.
2. The function uses global variables, which makes it less portable. The best practice would be to parameterize the function, where the configuration directory (currently represented as the global variable HPS_SERVICE_CONFIG_DIR) can be passed as an argument to the function.
3. It is recommended to add additional logging for each step of the process to provide clarity on the steps undertaken by the function and to assist with troubleshooting, if necessary.
4. Permissions of files should be checked and monitored on the fly. The function uses sensitive directories such as '/var/run/' and '/var/log/'. It should ensure the running script has necessary permissions for these operations.
5. All important output and return codes of each command should be trapped and logged to ensure traceability.
6. Any sensitive data such as passwords or keys, if any, should not be hardcoded in the scripts for security reasons. They should be fetched from secure sources or encrypted files.

2.17 count_clusters

Contained in `lib/functions.d/cluster-functions.sh`

2.17.1 Function overview

The function `count_clusters` is designed to count the cluster directories present in a base directory and output the count. The base directory is specified by the environment variable `HPS_CLUSTER_CONFIG_BASE_DIR`. If the base directory does not exist or no clusters are found within it, the function informs the user through a message output to stderr and returns zero.

2.17.2 Technical description

- **Name:** count_clusters
- **Description:** This function counts the cluster directories in a given base directory specified by the variable `HPS_CLUSTER_CONFIG_BASE_DIR` and outputs the count. If the base directory does not exist or there are no clusters, it outputs an error message to stderr and returns zero.
- **Globals:**
  - `HPS_CLUSTER_CONFIG_BASE_DIR`: This is the path to the base directory containing the clusters.
- **Arguments:** None.
- **Outputs:** If successful, reports the number of cluster directories present in the base directory. On failure (base directory doesn't exist or contains no clusters), prints an error message to stderr and returns `0`.
- **Returns:** Always returns `0`. The main purpose of the function is to output the count to stdout.
- **Example usage:**

```
- count_clusters
```

2.17.3 Quality and security recommendations

- Validation should be added to ensure that `HPS_CLUSTER_CONFIG_BASE_DIR` is not null or empty.
- The function's dependence on a global variable `HPS_CLUSTER_CONFIG_BASE_DIR` may be replaced with an argument, increasing reusability of the function and minimizing side effects.
- Consider providing more specific error messages for edge cases, such as when the path is not a directory or when the path is not permitted.
- For security, it'd be best to confirm the user has necessary permissions for the given path to avoid unexpected behavior.
- It might be beneficial to verify that each item in `clusters` is actually a directory, not a file.

- The script should make sure that the necessary commands, such as `shopt`, are available before calling them.
- The function's return value should reflect its success or failure. Currently, it always returns 0, which typically indicates successful execution.

2.18 `detect_storage_devices`

Contained in `lib/functions.d/storage_functions.sh`

2.18.1 Function overview

The function `detect_storage_devices` is used to detect all the storage devices in the system, gathering essential information about each device such as its model, vendor, serial, type, bus, size, usage, and speed. The list of devices and their data are formatted into a string, which is printed out by the function. This function relies on several helper functions to fetch specific information about each device.

2.18.2 Technical description

- **name:** `detect_storage_devices`
- **description:** Fetches a list of all storage devices present on the system using the helper function `get_all_block_devices`. For each device, the function gets details including model, vendor, serial number, type, bus, size, usage and speed using respective helper functions like `get_device_model`, `get_device_vendor`, etc. It formats these details into a string, which is then outputted.
- **globals:** None
- **arguments:** None
- **outputs:** Prints a string with details of all the storage devices. Each device's detail includes the device name (device), model, vendor, serial number, type, bus, size, usage, and speed.
- **returns:** None
- **example usage:** `detect_storage_devices`

2.18.3 Quality and security recommendations

- Check and handle possible errors in calls to helper functions for fetching device info. This would improve robustness.
- Escape possible special characters in device names and other strings to prevent unwanted interactions with the scripting language.
- Store all sensitive information, such as serial numbers, safely. This information should not be displayed in a context where unauthorized users could gain access to it.

- Consider using more specific variable names instead of general ones like `output` and `devs` to improve readability and maintainability of the code.
- Before calling helper functions to get device details, check if a device exists; this helps to prevent potential depending/command errors.
- Include logging at every step, especially where there are chances of failures, to track and rectify issues.

2.19 download_iso

Contained in `lib/functions.d/iso-functions.sh`

2.19.1 Function Overview

The `download_iso` function is designed to download ISO files for a given OS variant from a manufacturer's website. The function handles the creation of the required directory structure, constructs the URL for the ISO based on the given parameters, checks if the ISO is already present in the specified directory, and attempts to download the file if not present. Successful downloads are confirmed with a success message, while failures are flagged with an error message prompting to try again.

2.19.2 Technical Description

- **Name**: `download_iso`
- **Description**: This function is used to download ISO files for a specified OS variant onto the local system.
- **Globals**: None
- **Arguments**:
  - `$1` (cpu): The architecture of the CPU (e.g., `x86_64`).
  - `$2` (mfr): The manufacturer (e.g., `rockylinux`).
  - `$3` (osname): The name of the operating system.
  - `$4` (osver): The version of the operating system.
- **Outputs**: Echo commands provide information about the download process, such as whether the ISO file is already present or whether the download succeeded.
- **Returns**:
  - `0` if the target ISO file already exists or if the download is successful.
  - `1` if an unsupported OS variant is entered or if downloading the ISO file fails.
- **Example Usage**:

```
download_iso "x86_64" "rockylinux" "Rocky" "8"
```

2.19.3 Quality and Security Recommendations

1. Add more comprehensive error handling and messaging for different possible failure points (e.g. unsupported CPU types, unavailable URL, network errors).

2. Instead of hardcoding a base URL, consider an external configuration file or environment variables.
3. Restrict the rights of the target directory and the downloaded ISO files to limit potential security risks.
4. Consider additional verifications after the download. (e.g., checksum verification to ensure the integrity of the downloaded file)
5. Include comments describing each part of the function, making it easier for others to understand how the function works.

2.20 export_dynamic_paths

Contained in `lib/functions.d/system-functions.sh`

2.20.1 Function overview

The `export_dynamic_paths` bash function is designed for managing cluster configurations in a directory-based manner. The function prioritizes the retrieval and setting of cluster-specific configurations from a specified cluster or, if none is specified, from the currently active cluster. The function primarily modifies global environment settings to adjust to the proper configuration directory paths.

2.20.2 Technical description

- **Name:** `export_dynamic_paths`
- **Description:** This function exports dynamic variable paths based on a specified cluster name. If no name is specified, it retrieves the name of the currently active cluster. It sets the global variables `CLUSTER_NAME`, `HPS_CLUSTER_CONFIG_DIR` and `HPS_HOST_CONFIG_DIR` to their respective paths.
- **Globals:** [`HPS_CLUSTER_CONFIG_BASE_DIR`: Path to the base directory where cluster configurations are stored (default: /srv/hps-config/clusters)]
- **Arguments:** [`$1`: Name of the specific cluster to use; defaults to the active cluster if not specified]
- **Outputs:** While this function doesn't print to stdout, it does output error messages to stderr if no active cluster is found and none is specified.
- **Returns:** Returns `0` if the function execution completes successfully. If no active cluster is found and none is specified, it returns `1`.
- **Example usage:**

```
export_dynamic_paths "my-cluster" # Example of specifying a cluster
export_dynamic_paths              # Example of using the active
 ↪  cluster
```

2.20.3 Quality and security recommendations

1. Make sure to properly clean and validate the input to prevent arbitrary path vulnerabilities.
2. Add more error checking for different corner cases.
3. In cases where no active cluster exists and one isn't specified, consider whether failing quietly (i.e., not exporting any variables) might be a better approach than loudly (i.e., outputting an error and returning 1). This will depend on how your script uses this function.
4. Consider breaking up functionality into smaller functions to improve readability and ease of testing. For example, retrieving the name of the currently active cluster could be its own function.
5. Document the assumption that cluster configuration directories will have hosts subdirectories. Any deviations from this structure can cause problems.
6. Always verify the existence of directories before using them.
7. Avoid storing sensitive information, such as configuration files, in predictable and globally accessible locations.

2.21 `extract_iso_for_pxe`

Contained in `lib/functions.d/iso-functions.sh`

2.21.1 Function overview

The bash shell function, `extract_iso_for_pxe`, controls the extraction process of ISO files relevant for PXE (Preboot Execution Environment). It employs a series of local variables and decision constructs to determine if the ISO file exists, if it has been previously extracted, and to handle the extraction process.

2.21.2 Technical description

Name: `extract_iso_for_pxe`

Description: This function is designed to extract an ISO file for utilization in PXE. It first checks if the ISO file exists and whether it has been extracted before. If the necessary conditions are met, it extracts the ISO to a specified directory and then validates the extraction process.

Globals: `HPS_DISTROS_DIR: This global directs to the directory where distributions are located.`

Arguments: `$1: The CPU identifier. $2: The Manufacturer identifier. $3: The Operating System name. $4: The Operating System version.`

Outputs: It delivers textual output to describe the process' successes or failures, with details about the ISO location or potential issues.

Returns: - 1 if the ISO file is not found or there is a failure in the extraction process. - 0 if the ISO file has already been extracted, or the extraction process was successful.

Example usage:

```
extract_iso_for_pxe 'i386' 'HP' 'Ubuntu' '18.04'
```

2.21.3 Quality and security recommendations

1. A check should be included to validate the incoming arguments to avoid unintended behavior or errors.
2. The use of quotation marks for variable expansion, such as `"$iso_file"`, is a good practice for preventing word splitting and pathname expansion. It should be used consistently in the entire script.
3. Provide a more detailed error message if the iso file not found to help the user in debugging.
4. Instead of using `echo` for standard error redirection (>&2), consider using `printf` which is safer and more portable.
5. For an extra layer of security, input validation could be enhanced, verifying that the supplied `cpu`, `mfr`, `osname` and `osver` conform to expected formats.
6. It may be beneficial to include logging at different steps, so that in the event of failure, a log file can be consulted to identify the issue.

2.22 `extract_rocky_iso_for_pxe`

Contained in `lib/functions.d/iso-functions.sh`

2.22.1 Function overview

The function `extract_rocky_iso_for_pxe` is designed to extract an ISO file of a specific version of Rocky Linux for PXE. It takes in three parameters: the path of the ISO file, the version of the Linux distribution, and the CPU architecture. This function creates a directory based on the parameters given, then attempts to extract the contents of the ISO file into this new directory. If neither `bsdtar` nor `fuseiso` commands are available, it will output an error message and return a non-zero exit status.

2.22.2 Technical description

- **Name:** `extract_rocky_iso_for_pxe`
- **Description:** This function extracts an ISO for PXE given the location of the ISO, its version, and CPU architecture type. It primarily makes use of bsdtar and fuseiso to perform the extraction.

- **Globals:** HPS_DISTROS_DIR: The base directory where the content is extracted.
- **Arguments:**
  - $1: The location of the ISO file to extract.
  - $2: The version of the Linux distribution.
  - $3: The type of CPU architecture.
- **Outputs:** Prints out information on the extraction process and the location the content is extracted to. In case of errors, it prints out an appropriate error message.
- **Returns:** Returns 1 if either bsdtar or fuseiso are not found, otherwise nothing.
- **Example usage:** `extract_rocky_iso_for_pxe "/path/to/iso" "8.4" "x86_64"`

2.22.3 Quality and security recommendations

1. The function should check the existence of the provided ISO path before attempting extraction.
2. It should validate the input arguments to prevent potential code vulnerabilities.
3. The code might use a switch case instead of multiple `if` conditions to choose the extraction program.
4. To improve execution transparency, consider using standardized logging methods instead of `echo`.
5. Additionally, the function should handle potential cleanup upon encountering an error to leave the system in a clean state.
6. Make the function more secure by providing file and directory permissions when they're created.
7. Use absolute paths when performing operations to avoid relative path vulnerabilities.

2.23 fetch_and_register_source_file

Contained in `lib/functions.d/prepare-external-deps.sh`

2.23.1 Function overview

This function, `fetch_and_register_source_file`, is primarily used for fetching a source file from an input URL and registering it with a specific handler. It takes in three parameters: a URL, a handler, and an optional filename, which is obtained by extracting the base name from the URL if it's not supplied. The function fetches the source file by calling the `fetch_source_file` function with the URL and filename as arguments. If that's successful, it then proceeds to register this source file by calling `register_source_file` with the filename and handler as arguments.

2.23.2 Technical description

- **Name:** fetch_and_register_source_file
- **Description:** This function fetches a source file from a given URL and registers it with a specified handler. If the download is successful, it proceeds to registration.
- **Globals:** None
- **Arguments:**
  - `$1 (url)`: The URL from where the source file is fetched.
  - `$2 (handler)`: The handler with which the source file is registered.
  - `$3 (filename)`: An optional argument. When not provided, the base name from the URL is used as the filename.
- **Outputs:** The function doesn't directly produce any output. However, the `fetch_source_file` and `register_source_file` functions called by it may produce output.
- **Returns:** If the fetch operation fails, the function returns false. If the fetch operation succeeds, the function calls `register_source_file` and returns its return value.
- **Example usage:**

```
fetch_and_register_source_file "http://example.com/file.txt" "handler_name"
```

2.23.3 Quality and security recommendations

- **Error Handling and Reporting:** There should be handling for cases when the provided URL is not a valid URL or when the URL, handler or filename does not exist or are not accessible. Also, it would be beneficial to add informative messages for the user in case something goes wrong.
- **Input Validation:** Consider validating the URL before using it, checking for potential security issues like some form of injection or files that are too large.
- **Output validation:** The outputs of the `fetch_source_file` and `register_source_file` functions should be validated.
- **Encryption:** If the function is used for transferring sensitive data, encryption should be enforced during the fetch operation. It should also confirm the integrity of the downloaded files, possibly through checksums or digital signatures.
- **Data privacy:** If user data is processed, the function should respect the privacy of users and follow the data protection laws. Besides, personal identifiers should be sufficiently anonymized during the processing.
- **Documentation:** It would be helpful to have more detailed comments in the function to help other developers easily understand its purpose and functionality.

2.24 fetch_source_file

Contained in lib/functions.d/prepare-external-deps.sh

2.24.1 Function overview

The fetch_source_file function is built to download a file from a provided URL and store it in a destination directory. If no filename is given, it deduces the filename from the URL. It first checks if the file already exists in the destination directory, if so it skips the download process. If the file does not exist, it attempts to download it, and provides relevant feedback on the process.

2.24.2 Technical description

fetch_source_file is defined as follows: - **name**: fetch_source_file - **description**: Downloads a file from the input URL and stores it into a defined directory. If the file is already present, downloading is skipped. - **globals**: [HPS_PACKAGES_DIR: Directory where the files are downloaded. Default is /srv/hps-resources/packages/src] - **arguments**: [$1: URL to download the file from, $2: Name of the file to download. This value is optional, and if not provided, the name is inferred from the URL] - **outputs**: Status of the download operation (successful, already exists, or failed) - **returns**: Download status represented by Boolean values (0: Success or file already exists, 1: Fail) - **example usage**: fetch_source_file "http://example.com/file.tar.gz" "testfile.tar.gz"

2.24.3 Quality and security recommendations

1. Use different variables or add validation checks to avoid the possibility of variable overlap, ensuring that the url and filename variables are valid.
2. Verify the successful creation of the directory prior to attempting the download operation.
3. Add timeout to the CURL request to prevent the script hanging indefinitely if the URL is inaccessible.
4. Make use of secured protocols (HTTPS) to download files, this ensures the integrity and confidentiality of the downloaded files.
5. Ensure that the file successfully downloaded is as expected, via checksumming or other verification process.
6. Handle errors appropriately, not just echoing to stderr. This can be done with a custom error function or dedicated error handling segments.

2.25 generate_dhcp_range_simple

Contained in lib/functions.d/network-functions.sh

2.25.1 Function overview

The generate_dhcp_range_simple function is a Bash script designed to generate a range of IP addresses within a specified network, usually for DHCP (Dynamic Host Configuration Protocol) use. The function utilises the ipcalc utility to extract the network and broadcast range of the specified network.

2.25.2 Technical description

Function: generate_dhcp_range_simple()

- **Name**: generate_dhcp_range_simple
- **Description**: This function is used to generate a range of IP addresses in a certain network. It does so by using a network CIDR block and gateway IP as inputs, as well as an optional count for the range.
- **Globals**: None
- **Arguments**:
  - $1: network_cidr - A network CIDR block (e.g. 192.168.50.0/24)
  - $2: gateway_ip - An IP address for the network's gateway (e.g. 192.168.50.1)
  - $3: count - Optional argument. Specifies the number of IP addresses to include in the range. If not specified, a default value of 20 is used.
- **Outputs**: The function generates a list of IP addresses, which can be used as a DHCP range.
- **Returns**: The function echoes the range of IP addresses.
- **Example usage**: generate_dhcp_range_simple "192.168.50.0/24" "192.168.50.1" 25

2.25.3 Quality and security recommendations

1. Including input validation to ensure that the network CIDR block, gateway IP, and count (if specified) are in the correct format would improve function quality.
2. The use of a dedicated IP address manipulation library or utility would improve the function's reliability and accuracy.
3. The script should check that ipcalc utility is available in the system before execution. If it's not, it should provide a meaningful error message.
4. Consider handling edge cases such as network CIDR blocks that don't have a suitable range for the specified count.
5. Implement error handling to deal with potential issues that may arise during calculation (e.g., inability to parse the network CIDR block or gateway IP, failure to convert IPs to integers, etc).

6. To improve security, sanitize all inputs to avoid potential code injection attacks.

2.26 generate_ks

Contained in `lib/functions.d/kickstart-functions.sh`

2.26.1 Function overview

The `generate_ks` function is primarily used for generating kickstart configurations for a given host type and macid. It logs information about the function calls as well as requests for kickstart while setting up plain CGI headers. Configurable host variables are made available for the installer script including important network settings such as IP, netmask, hostname etc. It sets the state of the host configuration to "INSTALLING" and offers the script for the host installation.

2.26.2 Technical description

- **Name:** `generate_ks`
- **Description:** Generates a kickstart configuration for a given host type and macid (MAC ID), sets up required host variables, sets the state of the host configuration to "INSTALLING" and logs information.
- **Globals:**
  - `macid`: Contains macid (MAC ID) passed as argument.
  - `HOST_TYPE`: Contains host type passed as argument.
- **Arguments:**
  - `$1`: macid (MAC ID)
  - `$2`: Host type
- **Outputs:** Logs information about function calls and kickstart requests. Prints the contents of the installation script after rendering the template.
- **Returns:** Does not return a value
- **Example usage:**

```
generate_ks "macid123" "host_type123"
```

2.26.3 Quality and security recommendations

- It might be worth considering the return of values or exit statuses after critical steps, e.g., after retrieving the host configuration.
- An explicit error handling should be added, so that steps with potential failure (such as loading the host configuration) return an error message and an appropriate status code.
- Logging should be improved. All the activities within the function should be logged, not just when the function has been called.

- Some variables are set but never used (commented out). These variables should be removed if not required.
- Sensitive data such as IP and MAC IDs should be handled securely and stored encrypted if needed.
- The function could benefit from a more defensive programming style (for instance, validating arguments before use).
- The use of globals is not recommended because it allows for variables that can be modified by any part of the program. It's better to replace globals with function arguments or return values if possible.
- The function should be refactored to make it less complex and easier to maintain.

2.27 get_active_cluster_file

Contained in `lib/functions.d/cluster-functions.sh`

2.27.1 Function Overview

The `get_active_cluster_file` function returns the contents of the active cluster config file in Bash programming. This function primarily works by linking to the active cluster config directory and checking if it contains a file named `cluster.conf`. If so, it returns the contents of that file. If not, it will return an error.

2.27.2 Technical Description

- **name:** `get_active_cluster_file`
- **description:** This function is used to get the content of the active cluster config file. It checks if the symlink to the active cluster directory exists then it verifies if the symlink resolves to an existing file named `cluster.conf`. If these conditions are satisfied, it returns the content of `cluster.conf`.
- **globals:** [`HPS_CLUSTER_CONFIG_DIR`: This is the directory of the active cluster configuration]
- **arguments:** [`None.`]
- **outputs:** Either outputs the contents of the active cluster config file, or an error message if the file could not be found or read.
- **returns:** It returns `1` if an error occurred. Otherwise, it doesn't return a specific value as it prints the output directly.
- **example usage:**

  ```
  get_active_cluster_file
  ```

2.27.3 Quality and Security Recommendations

- Replace the hardcoded cluster.conf file name with a configurable variable.
- Include more specific error handling for each of the different possible points of failure.
- Regularly update the permissions on `cluster.conf` to ensure it can't be modified by unauthorized users.
- Guarantee the security of `HPS_CLUSTER_CONFIG_DIR`, preventing unauthorized access and movement.
- Consider using more reliable methods such as `pushd` and `popd` instead of `cd` for directory changes to ensure path integrity.
- Always clean or sanitize outputs, particularly if they're being used in subsequent scripts or commands.
- Avoid showing full directory paths in error messages. This can potentially expose sensitive information about the server.
- Verify that $link is a symlink to a file, not to a directory or other type of file.

2.28 `get_active_cluster_filename`

Contained in `lib/functions.d/cluster-functions.sh`

2.28.1 Function overview

The `get_active_cluster_filename` function is used to find and return the path to the `cluster.conf` file in an active cluster directory.

This Bash function primarily makes use of the `readlink` utility to retrieve the target of a symbolic link to an active cluster.

2.28.2 Technical description

- **Name:** `get_active_cluster_filename`
- **Description:** This function returns the path to `cluster.conf` in the directory of the active cluster. It first checks if a symbolic link to the active cluster exists. If it does, it then resolves the full path to the target of the symbolic link and appends `cluster.conf` to generate the full path of interest.
- **Globals:**
  - `HPS_CLUSTER_CONFIG_BASE_DIR: The base directory for the cluster configuration.`
- **Arguments:**
  - None
- **Outputs:**

  - The fully-qualified path to cluster.conf within the active cluster directory.
  - Error messages, when the symbolic link to an active cluster does not exist or cannot be resolved.

- **Returns:**

  - The function outputs a non-null status if the symbolic link does not exist or the target of the symbolic link cannot be resolved. Otherwise, it returns a null status.

- **Example Usage:**

```
get_active_cluster_filename
```

2.28.3 Quality and security recommendations

- For this function, it's highly advised to validate before using the HPS_CLUSTER_CONFIG_BASE_DIR environment variable, checking if it indeed corresponds to an existing directory.
- Keep the permissions of the file `cluster.conf` and its directories restricted to avoid unauthorized access or modifications.
- It's better to suppress the possible error message thrown by the `readlink` command to avoid exposure of sensitive system information.
- The function does not handle cases where `cluster.conf` does not exist in the directory. Adding a secondary check to verify the existence of `cluster.conf` before returning the path could reduce the possibility of file not found errors downstream.

2.29 get_active_cluster_info

Contained in `lib/functions.d/cluster-functions.sh`

2.29.1 Function overview

The `get_active_cluster_info` function in Bash, primarily, retrieves the information of the active cluster in a directory of clusters. The function evaluates the condition if there is only one cluster, if the active cluster link exists and if multiple clusters exist in the directory. Furthermore, it handles cases where no cluster directories are found or when a cluster configuration does not exist in the active cluster. It can be particularly utilized where a need to systematically read through multiple directories for certain information is required.

2.29.2 Technical description

- Name: `get_active_cluster_info` **- Description:** The function navigates a cluster directory, evaluates the status of clusters (viz. single, multiple, active, none, etc.), and attempts to output the configuration file path for an appropriate cluster accordingly. **- Globals:** * `VAR:HPS_CLUSTER_CONFIG_BASE_DIR`: This variable sets the base directory for the cluster configuration. If not set, defaults to `/srv/hps-config/clusters`. **- Arguments:** None **- Outputs:** The function primarily outputs the path to the configuration file of, respectively, the single cluster found, the active cluster, or the chosen one amongst multiple clusters. **- Returns:** Returns 1 if no cluster directories are found or if an active cluster configuration is not found. **- Example usage:**

```
get_active_cluster_info
```

This function does not take any arguments.

2.29.3 Quality and security recommendations

- It is suggested to have error handling capabilities for when the base directory does not exist or is not accessible.
- Verification of existence and readability of the selected configuration file prior to echoing the path could be beneficial.
- Introduce logging for activity monitoring and better error tracing. This ensures privacy practices and alerts on unexpected activities.
- Use of unquoted variables could lead to word splitting vulnerabilities. It is recommended to always quote your variables in Bash.
- It is essential to validate and sanitize all inputs into functions to ensure that correct and safe data is being used.
- Implement access controls to ensure that only authorized personnel can access, modify, or delete essential data.

2.30 `get_all_block_devices`

Contained in `lib/functions.d/storage_functions.sh`

2.30.1 Function overview

The `get_all_block_devices` function is a Bash function that fetches all block devices from the `sys/block` directory in a Unix-based system. For each device, it obtains the device name, checks whether the device type is 'disk', and if so, it prints this device name. The function excludes non-disk types such as loop, md, dm, and others.

2.30.2 Technical description

- **Name:** get_all_block_devices
- **Description:** This function iterates through all the files in the `/sys/block/` directory. For each file, it verifies if the device type is 'disk'. If it is, it prints the device name.
- **Globals:**
  - `devname` : It stores the name of the device file.
- **Arguments:**
  - No Arguments are needed for this function.
- **Outputs:**
  - It outputs the names of block devices where the device type is 'disk'.
- **Returns:**
  - The function does not explicitly return a value, but it results in having the disk type block devices' names printed.
- **Example usage:**
  - `get_all_block_devices`

On executing this function, it will print all the block device names where the device type is 'disk'.

2.30.3 Quality and security recommendations

1. **Defensive Programming**: Check if `/sys/block/` directory exists before proceeding. It will avoid possible errors.
2. **Error Handling**: Handle possibilities where `get_device_type` function call may fail.
3. **Security**: Avoid printing the device names directly. Depending on the context, disclosing those can be a security risk.
4. **Improving Readability**: Comment each section of code so that the function becomes self-explanatory.
5. **Follow a coding style guide**: Consistency could make the code easier to understand or maintain.

2.31 `get_client_mac`

Contained in `lib/functions.d/network-functions.sh`

2.31.1 Function overview

The `get_client_mac` function is designed to retrieve and output the MAC address of a specified client in a network based on its IP address. This function ensures the validness of the stated IP, triggers an Address Resolution Protocol (ARP) update, as well as uses both the `ip neigh` command and `arp -n` command to get and print the MAC address.

2.31.2 Technical description

```
- Name: get_client_mac
- Description: This function retrieves the Media Access Control (MAC) add
- Globals: None.
- Arguments:
  - $1: IP address of the client whose MAC address needs to be retrieved.
- Outputs: Prints the MAC address of the specified client based on its IP
- Returns: 1, if the IP address is not valid.
- Example usage: get_client_mac 192.168.1.2
```

2.31.3 Quality and security recommendations

- Implement proper input validation. This code assumes that the input passed is an IP address, but there's no validation to check if the format is correct.
- Check for the existence of required utilities like `ping`, `ip` and `arp`. The code doesn't handle scenarios where these utilities might be absent on the system.
- Be caution about executing `ping` or any other system commands directly from the function. It might open up attacks through command injection.
- Be aware of privacy concerns. Fetching MAC addresses can be seen as intrusive, as MAC addresses are often used to uniquely identify devices on a network.
- Use functions or utilities provided by your language of choice to reduce reliance on system commands. This will also likely improve the robustness and efficiency of your code.
- Regular upgrades and patching of the underlying system are critical to ensure security.

2.32 `get_device_bus_type`

Contained in `lib/functions.d/storage_functions.sh`

2.32.1 Function overview

The `get_device_bus_type` function is a bash command that is used to determine the bus type of a given device. The function takes in a device as an argument and checks its file path to see if it matches the criteria for different bus types (NVMe, SSD, or HDD). If the device is an NVMe device, the function will directly output "NVMe". For devices that do not match the NVMe naming convention, the function will call a different function `get_device_rotational` with the device as an argument. If that function returns "0", indicating that the device is a non-rotational device, the bus type of the device is assumed to be "SSD". If the device is a rotational device, it is assumed to be "HDD".

2.32.2 Technical description

- **name**: `get_device_bus_type`
- **description**: A function to determine the bus type of a device. The possible bus types it can return are “NVMe”, “SSD”, or “HDD”.
- **globals**: None
- **arguments**: [`$1`: A string that represents the device path]
- **outputs**: The bus type of the device.
- **returns**: 0 if successfully executed, non-zero on error.
- **example usage**:

```
bus_type=$(get_device_bus_type "/dev/nvme0n1")
echo $bus_type # Outputs: NVMe
```

2.32.3 Quality and security recommendations

- There should be more rigorous error checking in place. For example, the function should check if the argument passed is a valid device path.
- The function doesn’t handle errors from the `get_device_rotational` function. It should include error handling mechanism for this function.
- The function could return an error when a device type is not recognized or the input is not expected, rather than making an assumption it is a HDD.
- Instead of directly echoing the string of device type, consider using return codes for different types of devices, which could make the function more usable programmatically.
- Make sure that the function has read access to the file path specified before attempting to execute operations on it.
- Avoid potential command injections by validating the input to the function.
- It should handle the case sensitivity of device names. For example, NVMe could be in lower-case as well.

2.33 `get_device_model`

Contained in `lib/functions.d/storage_functions.sh`

2.33.1 Function overview

The `get_device_model` function fetches and returns the model of the device specified. This function is useful for retrieving the model name for any block device in a Linux environment, and this information can be used for various purposes like device identification, logging or any other device-specific logic. As a fallback, it will return the string “unknown” when the model name cannot be obtained.

2.33.2 Technical description

- **Name:** `get_device_model`
- **Description:** This function fetches the model of a device specified by the user and returns it. If the model name cannot be obtained, it returns the string “unknown”.
- **Globals:** None
- **Arguments:**
  - `$1`: The block device for which the model name is to be fetched.
- **Outputs:** Prints model name of device or “unknown” if not fetch-able.
- **Returns:** Nothing
- **Example Usage:**
  - Command: `get_device_model /dev/sda`
  - Response: `ST1000DM003-1CH162` or `unknown`

2.33.3 Quality and security recommendations

- Always validate the argument passed to the function. It is a good practice to make sure the required number of arguments are provided, and they are of expected type or format.
- A more explicit error message could be returned instead of “unknown” to provide more information to the end-user in cases where the model name cannot be fetched.
- Redirecting errors to /dev/null can hide potential issues. It would be better to handle errors and give proper messages whenever something goes wrong.
- Always remember to quote your variables in bash to prevent word splitting and pathname expansion.
- Consider limiting the permissions of the file the function is defined in and mark it as readonly, to prevent unauthorized modifications.

2.34 `get_device_rotational`

Contained in `lib/functions.d/storage_functions.sh`

2.34.1 Function overview

The `get_device_rotational()` function is a bash function for determining the rotation status of a device. This function works specifically on Linux systems, fetching the rotation information directly from the system files. The result can indicate whether the specific device is rotational or not. This could be useful for operations that require specific measures or optimizations for rotational or non-rotational storage devices.

2.34.2 Technical description

- Name: `get_device_rotational`
- Description: This function determines whether a given block device is rotational (like hard disk drives) or non-rotational (like solid-state drives). It accesses a particular system file using the device name, attempting to read out a value ('0' or '1') that indicates the device's rotation status.
- Globals: None
- Arguments:
  - `$1: dev` - a string specifying the name of the device for which to fetch the rotational status.
- Outputs:
  - '0' indicates the device is non-rotational.
  - '1' indicates the device is rotational.
- Returns: The rotation status of the input device.
- Example Usage: `get_device_rotational sda # Output: 1`

2.34.3 Quality and Security Recommendations

1. Implement error handling for cases where the input argument does not meet expected formats or when the system file does not exist for the specified device.
2. Use clear and descriptive naming conventions for parameters and variables to improve code readability.
3. As the function deals with system-level information, ensure that it is invoked by users with appropriate permissions. Unauthorized usage should be avoided.
4. Include input validation to check that the device exists and try to handle edge cases where the device may not be available.

5. Comment your code to make it easy for others (or future you) to understand what each part of the function is doing.
6. Sanitize system command (like `basename`) exaggerations and redirection to prevent potential command injection security weaknesses.

2.35 `get_device_serial`

Contained in `lib/functions.d/storage_functions.sh`

2.35.1 Function overview

The `get_device_serial()` function is a Bash method useful for querying hardware properties in Linux systems. It uses the `udevadm` command to fetch device properties,

specifically the `ID_SERIAL` property, which is the unique identifier for a device. This function receives a device name as input and returns the corresponding serial number. If it cannot find the serial number, it outputs “unknown”.

2.35.2 Technical description

Name:
`get_device_serial`

Description:
This Bash function is designed to extract a hardware device’s unique identifier (serial number) in Linux systems. It’s important in scenarios where it’s necessary to programmatically identify specific hardware for any task. The `udevadm` command is used alongside `grep`, `cut`, and `echo` utilities for finer manipulation of the retrieved device properties.

Globals:
None.

Arguments:
- `$1`: Device name. The hardware device to query the serial number for.

Outputs:
The function outputs the hardware device’s serial number. If the serial number cannot be located, it outputs “unknown”.

Returns:
No value is returned.

Example usage:

```
get_device_serial /dev/sda
```

2.35.3 Quality and security recommendations

1. Input sanitization: Add checks to validate that the input is a valid device path before proceeding.
2. Error handling: Rather than simply outputting “unknown”, consider adding more robust error handling to provide diagnostic information in case of execution errors.
3. Silent failure: Provide a fallback or a warning if `udevadm` is not present/installed in the system.
4. Command injection: Even though this function is probably safe from command injection vulnerabilities (assuming untrusted input can never be a device path), it’s good practice to ensure script variables are only used in safe contexts.

2.36 `get_device_size`

Contained in `lib/functions.d/storage_functions.sh`

2.36.1 Function overview

The function `get_device_size` is used in Bash to retrieve the size of specified device. It takes as input the device name, then uses the `lsblk` command to determine and return the size of the device. If the size cannot be obtained, it will return “unknown”.

2.36.2 Technical description

- **Name:** `get_device_size`
- **Description:** The function queries the size of the device specified by the user. If the size is not assessed, presumed as unknown.
- **Globals:** None
- **Arguments:**
  - `$1:` The name identifier of the device (`dev`) whose size is to be determined.
- **Outputs:** The size of the specified device or the string “unknown” if the size could not be determined.
- **Returns:** Returns 0 on successful execution of the command, and values > 0 on error conditions.
- **Example usage:**

  ```
  get_device_size /dev/sda
  ```

  This command will output the size of the device `/dev/sda`.

2.36.3 Quality and security recommendations

- Ensure permission checks are in place before executing the function as it interfaces with the system hardware which if accessed without correct permissions can lead to unauthorized information disclosure.
- Validate the input to confirm that the device indeed exists before trying to get its size. This will prevent unnecessary error statements and possible exposure of sensitive information.
- Use error handling to deal with any possible issues that might occur when executing `lsblk`.
- Employ a more precise error message instead of “unknown”, as it may benefit the debugging process and the user’s understanding of the outcome.

2.37 `get_device_speed`

Contained in `lib/functions.d/storage_functions.sh`

2.37.1 Function overview

The get_device_speed function is a Bash function that tests the read speed of a given device. It uses the dd command to read data from the device and direct it to null, then uses grep to extract the speed from the output. If the dd command fails or no speed can be determined, the function echoes “N/A”.

2.37.2 Technical description

- **Name:** get_device_speed
- **Description:** This function tests the read speed of a given device by utilizing the dd command to read data and direct it to null. The speed is then extracted from the output. If the process fails or there is no speed, the function returns “N/A”.
- **Globals:** None.
- **Arguments:**
  - **$1:** dev - The name of the device whose speed is being checked.
- **Outputs:** The function will output the speed of the device being tested in MB/s. If this cannot be determined, it will output “N/A”.
- **Returns:** None.
- **Example usage:**

```
get_device_speed "/dev/sda"
```

This will output the speed of the device “/dev/sda” or “N/A” if it cannot be determined.

2.37.3 Quality and security recommendations

1. The function should include input validation to ensure the device exists before attempting to read its speed.
2. The function currently uses dd which can be dangerous if used incorrectly. Consider alternative methods of determining device speed that avoid the risk of data loss.
3. Consider adding error handling to catch potential issues during the execution of the dd or grep commands.
4. If utilized in a script with other functions, consider checking for any potential collisions with the local variable dev.
5. It may be beneficial to refactor the function to return an error status code if the speed cannot be determined rather than echoing “N/A”.

2.38 get_device_type

Contained in lib/functions.d/storage_functions.sh

2.38.1 Function Overview

This function, named `get_device_type`, is responsible for identifying the type of a given device in a Unix-based system. It uses the udevadm utility to query the property of the specified device and parses the output to extract the device type. If the device type cannot be located (due to either the device not being recognized or an error occurring), the function will default to identifying the device as a "disk".

2.38.2 Technical Description

- **Name**: get_device_type
- **Description**: This function is used to determine the type of a specified device within a Unix-based environment. It will return "disk" as a default output if the device type cannot be determined.
- **Globals**: None
- **Arguments**:
  - $1: `dev` – The device for which the type is to be determined.
- **Outputs**: Writes the device type to stdout. If the device type cannot be specifically determined, it outputs "disk".
- **Returns**: The function itself does not explicitly return a value.
- **Example Usage**:

```
# Get the device type for /dev/sda
device_type=$(get_device_type /dev/sda)
echo $device_type
```

2.38.3 Quality and Security Recommendations

- Add error handling that will inform the user if the entered device is invalid or cannot be located.
- Update the function documentation to detail any assumptions or limitations that the function has. In particular, clarify what types of devices are recognized, and what types of devices will default to "disk".
- Since the function pipes the output of a command to grep, make sure that the context in which function is running trusts that environment. In untrusted environments, the function could be tricked into running unexpected commands.
- Consider changing the function to exit with an error code, instead of defaulting to "disk", if the device type cannot be determined.
- Favour the use of grep -q or grep –quiet when possible to reduce unnecessary data written to stdout.
- Make certain that the utility udevadm and utilities used in this function (like grep and cut) are up-to-date to avoid any potential security issues.

2.39 get_device_usage

Contained in lib/functions.d/storage_functions.sh

2.39.1 Function overview

The get_device_usage function is a Bash shell function that retrieves the usage status of a specific device. This function works by extracting information from the lsblk command output to grasp the usage scenario of the specified device. This entire process is facilitated through a series of commands and piping outputs to manipulate the data into a suitable format before outputting it.

2.39.2 Technical description

Name:
get_device_usage

Description:
The function fetches and outputs the mount point statuses of a specified linked device. If there is no output or the device is not used, it outputs “unused”.

Globals:
None

Arguments:
- $1: dev - The specified device to check its mount point status.

Outputs:
If the device is used, it returns a comma-separated string representing the mount points where the device is being used. If the device is not used, it returns the string “unused”.

Returns:
The function does not have a specific return value as it echoes its result directly.

Example usage:

```
get_device_usage /dev/sda1
```

This example will output used mount points of the /dev/sda1 device.

2.39.3 Quality and security recommendations

1. Error Handling: Include error checking and handling mechanisms to deal with unexpected input or failures from the lsblk command.
2. Input Validation: Validate the device argument to ensure it’s a valid device before attempting to run lsblk.
3. Code Simplification: Seek ways to simplify the function for readability. This might mean breaking down the line containing lsblk, grep, tr and sed into more manageable sections.

4. Secure Handling of Variables: Use `$dev` inside double quotes in the `lsblk` command to prevent word splitting and filename expansion which might lead to potential command injection or unexpected behavior.
5. Quiet errors suppression: Consider adding `-e` and `-u` options to prevent the script from running with uninitialized variables and to make the script exit when a command fails. This will make debugging easier when something goes wrong.

2.40 `get_device_vendor`

Contained in `lib/functions.d/storage_functions.sh`

2.40.1 1. Function Overview

`get_device_vendor` is a bash function designed for identifying the device vendor. The function takes a device path as an argument and reads from the `/sys/block/<basename of the device path>/device/vendor` file to fetch vendor information. If it is unable to fetch the vendor information, it returns a string "unknown".

2.40.2 2. Technical Description

- **Name**: `get_device_vendor`
- **Description**: This function takes a device path as an argument in order to outsource the vendor of the provided device. In case the provided device doesn't have a vendor, the function will return a string `"unknown"`.
- **Globals**: None
- **Arguments**:
  - `$1: Device path from which the function will extract the basename and use in the fetch vendor process.`
- **Outputs**: The vendor of a given device or "unknown" if it's not found.
- **Returns**: The device vendor name or "unknown".
- **Example Usage**:

  ```
  get_device_vendor /dev/sda
  ```

  Will return the vendor of the device located at /dev/sda.

2.40.3 3. Quality and Security Recommendations

- **File Existence**: Before attempting to read the device/vendor file, check whether it exists to avoid unnecessary error handling.

- **Argument Validation**: The function doesn't validate the input argument before use. Implement input validation to ensure the provided path is a valid device path.
- **Error Messages**: Instead of routing all error messages to /dev/null, consider logging the actual error message for debugging purposes.
- **Return Status Codes**: For better error handling, the function should return distinct status codes for different error/results.
- **Input Sanitization**: Unfiltered user input can lead to potential security vulnerabilities. Always sanitize user inputs before using them in a function.

2.41 get_external_package_to_repo

Contained in `lib/functions.d/repo-functions.sh`

2.41.1 Function Overview

The Bash function `get_external_package_to_repo()` downloads a RPM package from a provided URL and places it within a specified repository directory. The function Authenticates the url and repo path, checks the existence of the repository and file type, and returns descriptive log messages and typical codes upon success or various errors.

2.41.2 Technical Description

Name: `get_external_package_to_repo()`

Description: This function downloads an RPM file from a given URL and saves it into a specified repository directory on a local system. It performs verification checks for the input parameters and the repository directory, ensures the URL points to an RPM file, and handles potential download errors.

Globals: None

Arguments: - `$1:` URL The URL pointing where the RPM package to be downloaded is located. - `$2:` `repo_path` The path to the repository directory where the RPM file will be saved.

Outputs: Log messages concerning the results of each command, whether it was successful or not and whether certain conditions were met.

Returns: - `0` if the function successfully downloaded the RPM file. - `1` if either the URL or the repository path were not provided. - `2` if the repository directory does not exist. - `3` if the URL does not point to a RPM file. - `4` if the download failed.

Example Usage:

```
get_external_package_to_repo "http://example.com/package.rpm" "/path/t
```

This would attempt to download the `package.rpm` file from `http://example.com/` and save it in `/path/to/repo`.

2.41.3 Quality and Security Recommendations

1. Implement validation to ensure that the URL is a HTTPS URL to provide an added layer of security.
2. Implement checksum validation to ensure the integrity of the downloaded file.
3. Implement user permissions restrictions for the target repository directory to ensure that unauthorized users cannot modify or delete any downloaded files.
4. Add more descriptive error messages to guide a user in troubleshooting.
5. Consider adding a check for free disk space before attempting download.
6. Add verification logic to ensure that the downloaded file doesn't already exist in the repository.
7. Implement a progress tracker to provide user feedback during large file downloads.
8. Add proper logging mechanism for accurate troubleshooting and record keeping.

2.42 `get_host_type_param`

Contained in `lib/functions.d/cluster-functions.sh`

2.42.1 Function overview

The `get_host_type_param` function is a Bash function that accepts two arguments and uses them to query a referenced associative array. Depending on the values of the arguments, it retrieves and echoes a value from the referenced array. If non-existent keys are queried, the function will echo an empty string.

2.42.2 Technical description

- **name**: `get_host_type_param`
- **description**: This Bash function accepts two arguments which represent the name of an associative array and a key. It uses them to reference the array and echo back the corresponding value. If the key doesn't exist in the array, the function will echo an empty string.
- **globals**: None
- **arguments**:
  - `$1`: This argument should be the name of an associative array which is to be directly referenced.
  - `$2`: This argument should be a key in the associative array defined by `$1`.
- **outputs**: Echoes the value corresponding to the key in the associative array or an empty string if the key is not found.
- **returns**: None

- **example usage**: `bash declare -A my_array=(["key"]="value") get_host_type_param "my_array" "key" # Returns: value get_host_type_param "my_array" "nokey" # Returns: (Empty string)`

2.42.3 Quality and security recommendations

1. For improved security, consider validating the input. Specifically, ensure the first argument is a declared associative array and the second argument is a string.
2. When declaring associative arrays, consider having a naming convention that differentiates them from normal variables. This way, it is easier to avoid programming errors where a normal variable is mistakenly passed as the first argument.
3. To avoid unexpected behavior, consider adding error handling that will inform the user if a key does not exist in the array.
4. Consider adding a check to ensure the referenced variable `ref` does not already exist in other parts of your code before declaring it. Overwriting variables could potentially result in bugs.

2.43 get_iso_path

Contained in `lib/functions.d/iso-functions.sh`

2.43.1 1. Function overview

The Bash function `get_iso_path` is used to generate a path to an ISO file within a directory specified by the `HPS_DISTROS_DIR` environmental variable. This function first verifies if `HPS_DISTROS_DIR` is set and is a directory. If these conditions are met, it appends "/iso" to `HPS_DISTROS_DIR` and prints the resultant string. Otherwise, it prints an error message and returns 1.

2.43.2 2. Technical description

```
get_iso_path() {
  if [[ -n "${HPS_DISTROS_DIR:-}" && -d "$HPS_DISTROS_DIR" ]]; then
    echo "$HPS_DISTROS_DIR/iso"
  else
    echo "[x] HPS_DISTROS_DIR is not set or not a directory." >&2
    return 1
  fi
}
```

- **Name**: `get_iso_path`
- **Description**: This function checks if the `HPS_DISTROS_DIR` variable is set and whether it indicates a valid directory. If so, it appends "/iso" to it and returns the resulting string. If not, it raises an error and returns the exit code 1.

- **Globals**: [HPS_DISTROS_DIR: The directory where the ISO files are stored.]
- **Arguments**: [None]
- **Outputs**: If successful, it outputs the path to an ISO file. If it fails, it sends an error message to stderr.
- **Returns**: It returns 0 if successful, otherwise it returns 1.
- **Example usage**: Not applicable, as the function does not take any arguments. It can be called in a script as `get_iso_path`.

2.43.3 3. Quality and security recommendations

For improving quality and security of the `get_iso_path` function, you can consider the following:

1. Add error handling and descriptive error messages.
2. Check if the directory contains any ISO files and raise an error accordingly.
3. Ensure that the HPS_DISTROS_DIR variable isn't injected maliciously by validating the path.
4. Consider hiding sensitive information from the error messages that could be used for malicious activities.
5. Keep the function updated with any new shell scripting best practices related to directory and path handling.
6. Use a standardized method for logging error messages.
7. Handle other potential issues like permission errors when accessing the directory.
8. Always test the function in different scenarios to identify any potential bugs or room for improvements.

2.44 `get_provisioning_node`

Contained in `lib/functions.d/configure-distro.sh`

2.44.1 Function Overview

The `get_provisioning_node` function provides an effortless way to retrieve the default gateway IP address in a Linux environment, also referred to as the provisioning node. This function utilizes the `ip route` and awk command-line utilities to extract and display the IP address of the default gateway.

2.44.2 Technical Description

`get_provisioning_node` This Bash function retrieves the default gateway IP address.

Globals: None

Arguments: None

Outputs: The default gateway IP address (provisioning node)

Returns: None

Example Usage:

```
get_provisioning_node
# Output: 192.168.1.1 (actual output differs based on network
 ↪  configuration)
```

This function doesn't take any input parameters and doesn't use global variables. When called, it displays the IP address of the default gateway.

2.44.3 Quality and Security Recommendations

- The function should handle the error case where "ip route" command is not available or the user does not have required permissions.
- Consider sanitizing the output of the `ip route` command to ensure it only returns valid IP addresses.
- It would be beneficial to provide a fallback option in case the IP command fails to return the expected result.
- The function does not currently return an exit code which can be a potential limitation if used within another script that depends on the success/failure of the function. To enhance usability, consider having the function return an exit code based on the success of the IP address retrieval operation.
- Use this function with caution as revealing the IP address of your default gateway could potentially expose your network to security risks. Therefore, it is recommended to use it within a secure and trusted environment.
- Lastly, always follow the principle of least privilege. Only grant permissions required for executing the function. This helps minimize any potential damage if the function is misused or mishandled.

2.45 handle_menu_item

Contained in `lib/functions.d/ipxe-menu-functions.sh`

2.45.1 Function Overview

The `handle_menu_item` function, as the name suggests, is used to handle various menu-related operations in BASH. This function receives two arguments; the first being the item (or the operation) to be performed and the second being the machine address (MAC). Subsequently, based on the type of item, this function performs a specific operation such as `init_menu`, `install_menu` etc.

Each menu item operation is nested inside a case block which allows the function to choose which operation to perform based on the argument received. The function

also provides for fail-safes and logs for each operation to track any potential errors or misconfigurations.

2.45.2 Technical Description

Below is a detailed technical breakdown of the function `handle_menu_item`:

- **Name:** `handle_menu_item`
- **Description:** This function is responsible for handling various menu-related operations based on the provided arguments.
- **Globals:** None.
- **Arguments:**
  - `$1: item` - Represents the specific operation to be performed.
  - `$2: mac` - Represents the MAC address.
- **Outputs:** Depends on the operation selected. However, in general, it could be logs, errors, initiation of other functions or changing states.
- **Returns:** None, i.e., null.
- **Example usage:** `handle_menu_item reboot 00:11:22:33:44:55`

2.45.3 Quality and Security Recommendations

- It might be beneficial to implement better error handling. Currently, the function only provides a message for an unknown menu item; however, improved error handling could capture a broader range of potential issues.
- The fail-safe `*)` could potentially be more informative about what menu items would be valid. Users might make a typographical error and not know what the correct options are.
- Avoid using echo for generating output. There are safer alternatives such as printf that better handle potential issues related to arbitrary content.
- Several sections of the function seem to contain hardcoded sections that could potentially be moved towards a more dynamic or data driven approach.
- Add input validation check at the beginning to ensure the menu item and Mac address values are not empty.

2.46 has_sch_host

Contained in `lib/functions.d/host-functions.sh`

2.46.1 Function Overview

The function `has_sch_host()` checks if there are any configuration files (`*.conf`) in the configuration directory (`HPS_HOST_CONFIG_DIR`) that are of type 'SCH'. If such a configuration file exists, the function succeeds, otherwise, it fails. If the configuration directory does not exist, an error message is output and the function returns failure.

2.46.2 Technical Description

- Name: `has_sch_host`
- Description: This function scans through the configuration directory for `.conf` files and checks if there is at least one file of type `'SCH'`. If such a file exists, the function returns success, otherwise it returns failure. If the configuration directory does not exist, an error message is output and the function returns failure.
- Globals:
  - `HPS_HOST_CONFIG_DIR`: This global variable is used to define the path of the host configuration directory that is to be searched.
- Arguments: None
- Outputs: Error message if the host config directory specified by `HPS_HOST_CONFIG_DIR` is not found.
- Returns:
  - `0` if at least one SCH type config file is found.
  - `1` if no SCH type config file is found or if the config directory is not found.
- Example Usage:

```
HPS_HOST_CONFIG_DIR="/path/to/config/dir"
if has_sch_host; then
  echo "SCH host found."
else
  echo "No SCH host found."
fi
```

2.46.3 Quality and Security Recommendations

1. Always enclose path variables in quotes to avoid issues with spaces or special characters in file paths.
2. It would be a good practice to confirm that the configuration files being searched are readable before running the `grep` command.
3. The script assumes that the script user has read permissions to all directories and files involved. A check should be implemented to ensure that the current user has the necessary permissions before executing the function.
4. Consider using more meaningful exit codes or provide them as constants at the beginning of the script for easier debugging.
5. Proper error handling can be done for the situation where the `HPS_HOST_CONFIG_DIR` variable is empty or not set.

2.47 host_config_delete

Contained in `lib/functions.d/host-functions.sh`

2.47.1 Function overview

The function `host_config_delete()` is used to delete a specific configuration file of a host determined by its MAC address. It first checks if the configuration file exists. If it does, it deletes the file and logs an informational message. If the file doesn't exist, it logs a warning message.

2.47.2 Technical description

- **Name:** `host_config_delete()`
- **Description:** The function deletes a host configuration file based on the provided MAC address.
- **Globals:** [HPS_HOST_CONFIG_DIR: This global is used to specify the directory of the host configuration files.]
- **Arguments:** [$1: mac, This argument specifies the MAC address of the host for which the configuration file needs to be deleted. $2: config_file, This argument specifies the configuration file to be deleted based on the MAC address.]
- **Outputs:** An informational message stating the host configuration file was deleted or a warning message if the configuration file is not found.
- **Returns:** Returns `0` if the configuration file is deleted successfully, or `1` if the configuration file doesn't exist.
- **Example usage:** `host_config_delete "12:34:56:78:9a:bc"`

2.47.3 Quality and security recommendations

1. Ensure that the `HPS_HOST_CONFIG_DIR` global is properly initialized and secured against unauthorized access.
2. Validate the `mac` parameter to prevent any sort of code injection.
3. Use a safer method to remove files other than `rm -f` to avoid accidental deletion of critical files.
4. Implement more robust error handling for cases where the file deletion fails for reasons other than non-existence.
5. Log both successful deletions and unsuccessful attempts with detailed messages in a dedicated and secure log system.

2.48 `host_config_exists`

Contained in `lib/functions.d/host-functions.sh`

2.48.1 Function overview

The `host_config_exists()` function is a BASH script utility designed to check whether specific host configuration files exist in a designated directory. It takes a Media

Access Control (MAC) address as the argument, forms a path to the should-be existing file, and checks if the .conf file indeed exists at the specified location.

2.48.2 Technical description

- **Name**: `host_config_exists`
- **Description**: This function checks if a configuration file named after the provided MAC address exists in the predefined host configuration directory.
- **Globals**: `HPS_HOST_CONFIG_DIR`: The directory where host configuration files are stored.
- **Arguments**:
  - `$1`: MAC address of the host whose configuration file's existence will be checked
- **Outputs**: None, the function doesn't produce any output.
- **Returns**: An exit statuses representing 'true' (if file exists) or 'false' (if file does not exist).
- **Example usage**:

```
if host_config_exists "MAC_ADDRESS_HERE"; then
  # Do something if the file exists
else
  # Do something if the file doesn't exist
fi
```

2.48.3 Quality and security recommendations

1. It's recommended that the content of the `HPS_HOST_CONFIG_DIR` variable is validated for proper format and security before this variable is put into use in the function to prevent potential directory traversal vulnerabilities.
2. You may want to use full paths to the commands (`[ [` and `-f`) to prevent potential issues with PATH hijacking.
3. Make sure that the MAC address format of the `$1` argument is validated before it's used in forming the `config_file` path.
4. This function currently doesn't handle the scenario of when the provided MAC address is empty. Error handling could be added to improve the robustness of this script.
5. Consider using more explicit variable names to increase readability.
6. You might want to return a standardized error code instead of relying solely on the exit status in the script to improve debugging information.

2.49 host_config

Contained in `lib/functions.d/host-functions.sh`

2.49.1 Function Overview

This function, called `host_config()`, is used to handle a configuration file for a specified host. The host is defined by its MAC address. The function can read the configuration file, check if a key exists, compare a key's value to a supplied value, and even set a new value for a key in both the running script and the physical configuration file.

The function is idempotent, meaning the config file is only read once during the execution of the script, regardless of how many times the function is called.

Particular actions to perform are determined by the command sent as the second argument, which can be 'get', 'exists', 'equals', 'set', or any other for an error message.

2.49.2 Technical Description

- **Name:** `host_config`
- **Description:** used to handle a host configuration defined by its MAC address in various ways.
- **Globals:**
  - VAR: Briefly describe the VAR global variable here
- **Arguments:**
  - `$1`: MAC address used to locate and identify specific host config.
  - `$2`: Command to execute on the configuration file (can be 'get', 'exists', 'equals', or 'set').
  - `$3`: Key to be manipulated or inquired about in the host configuration.
  - `$4`: (Optional) Value to be used in conjunction with the command argument.
- **Outputs:** Increments a counter `__HOST_CONFIG_PARSED` denoting that the config file was parsed. Writes to the host's config file.
- **Returns:** Exit status of the function. Returns 2 for an invalid command
- **Example Usage:**

```
host_config "01:23:45:67:89:ab" "set" "TIMEZONE" "UTC"
```

2.49.3 Quality and Security Recommendations

1. Code comments are clear and well-written. Continue this practice.
2. Consider implementing input validation for the MAC address and other arguments.
3. Ensure that permissions on the configuration file prevent unprivileged users from reading or altering it.
4. If the configuration file contains sensitive data, consider implementing encryption measures.
5. Error messages should ideally print to `stderr`, which is already done for invalid commands.
6. Provide a usage message when the function is called incorrectly. This could be embedded in the existing error message for invalid commands.

2.50 host_config_show

Contained in `lib/functions.d/host-functions.sh`

2.50.1 Function overview

The `host_config_show` function in Bash is primarily used to process a configuration file for a specific host identified by a MAC address. If a configuration file does not exist for the given MAC address, it logs the info and returns. If a configuration file does exist, it reads each line (considering each line as a key-value pair, separated by '=') and processes it by trimming and escaping special characters in the value associated with each key. After processing, it then echoes these key-value pairs.

2.50.2 Technical description

- **name**: host_config_show
- **description**: This function is designed to process a host's configuration file identified by its MAC address. It performs tasks such as reading key-value pairs, trimming and escaping special characters in the values, and returning the key-value pairs.
- **globals**: [
  - HPS_HOST_CONFIG_DIR: Directory where host configuration files are stored.]
- **arguments**: [
  - $1: The MAC address used to identify the host's configuration file.]
- **outputs**: It outputs processed key-value pairs read from the configuration file.
- **returns**: It returns 0 if no configuration file exists for the given MAC address.
- **example usage**:

```
host_config_show "00:0a:95:9d:68:16"
```

2.50.3 Quality and security recommendations

- Escape all the other special characters that can cause issues if not properly handled.
- Validate the input MAC address format.
- Implement error handling to manage scenarios when the directory does not exist or does not have the required permissions.
- Test the function with a large configuration file to ensure performance is not affected.
- When displaying log messages, consider using a logging level more granular than 'info', so that users can control the verbosity of the logs.
- If possible, manage the secured reading of configuration files, especially if they contain sensitive information.

- Consider refactoring the method's internals in a way that doesn't just echo out the output, in order to provide more flexible usage of the function.

2.51 host_initialise_config

Contained in lib/functions.d/host-functions.sh

2.51.1 Function overview

The function host_initialise_config is primarily used for setting up the host configuration. This function takes a local MAC address as an argument, generates a configuration file name using that MAC address and the host configuration directory ${HPS_HOST_CONFIG_DIR}. It then creates the mentioned directory, sets the state to "UNCONFIGURED", and logs the completion of the host config initialization process.

2.51.2 Technical description

- **Name:** host_initialise_config
- **Description:** This function is used to initialize the host configuration. The function creates the host configuration directory if it does not exist and sets the configuration's state to "UNCONFIGURED".
- **Globals:** [${HPS_HOST_CONFIG_DIR}: The directory for storing host configurations]
- **Arguments:** [$1: Local mac address to initialize host configuration, not optional]
- **Outputs:** Does not output any variables, but logs the completion of the initialization with the message "Initialised host config: (config_file)"
- **Returns:** No return information
- **Example usage:**

  ```
  host_initialise_config "C0:FF:EE:C0:FF:EE"
  ```

2.51.3 Quality and security recommendations

1. Always ensure that the MAC address argument is valid or sanitized before passing it to the function as it may lead to unhandled exceptions.
2. Add error checking logic and include exception handling for actions like non-existent directories or inaccessible file paths.
3. Ensure that the use of hps_log is secure and does not expose sensitive logs in insecure locations.
4. Uncomment the pieces of code related to the creation timestamp and refine in a way that doesn't cause an error. This will enhance traceability and aid in debug processes.

5. Explore mechanisms to return status or error codes to make the function more robust and usable in the script. The function currently does not return any value.
6. Consider encrypting sensitive data in the configuration file to enhance security.

2.52 host_network_configure

Contained in `lib/functions.d/host-functions.sh`

2.52.1 Function Overview

The function `host_network_configure()` is used to set up various network configurations based on input parameters and specified cluster configurations. The function takes two input arguments - `macid` and `hosttype` - and sets various local network details including network base, DHCP_IP, and DHCP_CIDR from the cluster configuration.

This function also performs various checks and logs appropriate messages in case of missing required parameters or unavailability of the needed command `ipcalc`.

2.52.2 Technical Description

- Name: `host_network_configure()`
- Description: This function is used to set up various network configurations for a host in a cluster.
- Globals: `macid`, `hosttype`, `dhcp_ip`, `dhcp_cidr`, `netmask`, `network_base`
  - `macid`: The MAC id of the network interface.
  - `hosttype`: The type of the host.
  - `dhcp_ip`: The IP address provided by DHCP.
  - `dhcp_cidr`: CIDR subnet mask provided by DHCP.
  - `netmask`: Network mask computed by `ipcalc`.
  - `network_base`: Network base address computed by `ipcalc`.
- Arguments: `$1`, `$2`
  - `$1`: The MAC id of the network interface.
  - `$2`: The type of the host.
- Outputs: Logs details of success and failure events.
- Returns: `1` when required parameters are missing or required command is not available, otherwise it doesn't explicitly return any value.
- Example usage: `host_network_configure "MAC_ID" "HOST_TYPE"`

2.52.3 Quality and Security Recommendations

- The function should return unambiguous values consistently. It does not explicitly specify a return on success, just failure.

- The function utilizes the `ipcalc` command, the absence of which triggers a non-zero exit code. This creates a dependency which should be clearly documented.
- Integrate input validation to ensure that the `macid` and `hosttype` arguments being passed to the function are well-formed to avoid undefined behavior.
- Log sensitivity data such as IP addresses, CIDR notations, netmasks etc. can lead to security risks. Always ensure that logs do not expose sensitive details unless absolutely necessary and approved.
- Employ permission checks to ensure that the script is not executed with higher than necessary privileges, which can potentially compromise security.
- The function should include more detailed error handling, including potential issues with sourcing `cluster_config`, and extraction of `dhcp_ip` and `dhcp_cidr`.

2.53 hps_log

Contained in `lib/functions.d/hps_log.sh`

2.53.1 Function overview

The `hps_log` function is designed to maintain the system log for the hypothetical product or system (hps). It accepts two parameters, namely level and raw message. The level indicates the severity or importance of the log entry, while the raw message is the actual content to be recorded in the log. The function also utilizes the environment variables `HPS_LOG_IDENT` and `HPS_LOG_DIR` which represent the identification tag for log messages and the path directory of the log file respectively.

2.53.2 Technical description

Here is the definition block for `hps_log`:

- **name**: `hps_log`
- **description**: Maintains a log of system activities in an organized manner. Uses a standardized logging format that includes a timestamp, identification tag, log level, and message content.
- **globals**:
  - `HPS_LOG_IDENT`: Default identification tag for log messages. Falls back to "hps" if not present.
  - `HPS_LOG_DIR`: Directory location for the systemic log file. Defaults to "/var/log" if unspecified.
- **arguments**:
  - `$1`: Log level to indicate the significance of the message.
  - `$*`: The actual content of the log message.
- **outputs**: The function writes the log entry into the designated log file.

- **returns**: No explicit return value.
- **example usage**:
  bash hps_log "INFO" "Application started successfully."

2.53.3 Quality and security recommendations

- It would be prudent to sanitize and validate the log message to prevent content that could potentially harm the system or compromise its security, e.g., script injection attempts.
- To improve file safety and integrity, permissions for the log file should be correctly set to write-only for the software and read-only for users as needed.
- It is best to have the log file rotated periodically to manage file size, storing older logs in an archive for later perusal if necessary.
- For better error handling, the function could signal a warning or error when it fails to write to the log file.

2.54 hps_services_restart

Contained in lib/functions.d/system-functions.sh

2.54.1 Function Overview

The function hps_services_restart() is used to restart all services under the control of the Supervisor process control system. The function first configures the Supervisor services using the configure_supervisor_services function. It then reloads the Supervisor configuration using the reload_supervisor_config function. Finally, it restarts all services by executing the supervisorctl restart all command with the path of the supervisord configuration file specified.

2.54.2 Technical Description

- Name: hps_services_restart
- Description: This function is utilized to restart all services managed by Supervisor. It achieves this by first configuring the Supervisor services, reloading the Supervisor configuration, and finally executing a command which restarts all the services.
- Globals: [HPS_SERVICE_CONFIG_DIR: This is a path of the directory containing the Supervisor configuration file to be used.]
- Arguments: None
- Outputs: Restarting of all Supervisor-managed services happens.
- Returns: Does not return anything explicitly.
- Example Usage:

```
hps_services_restart
```

2.54.3 Quality and Security Recommendations

1. Validate the existence of the HPS_SERVICE_CONFIG_DIR directory. If not present, the function should stop the execution and print an appropriate error message.
2. Gracefully handle potential errors during the execution of `configure_supervisor_services`, `reload_supervisor_config`, and `supervisorctl -c "${HPS_SERVICE_CONFIG_DIR}/supervi` `restart all` commands. If there's an error, the function should stop the execution and return or print an appropriate error message.
3. Set appropriate permissions to the `supervisord.conf` file to prevent unauthorized modifications.
4. Consider logging all the actions performed within this function for audit purposes. Log files should be protected against unauthorized access.
5. It is essential to verify if the user calling this function has the necessary permissions to restart the services to prevent misuse.

2.55 hps_services_start

Contained in `lib/functions.d/system-functions.sh`

2.55.1 Function overview

The `hps_services_start` function primarily serves to initiate all the services defined in the supervisor configuration by using the supervisor control (supervisorctl) command. It manages this through a clear pipeline of operations which starts with configuration set up and ends with all the defined services starting up.

2.55.2 Technical description

- **Name**: hps_services_start
- **Description**: A function that configures supervisor services, reloads the supervisor configuration, and starts all services defined in the supervisor configuration.
- **Globals**: HPS_SERVICE_CONFIG_DIR: Directory path to the supervisord configuration file.
- **Arguments**: None
- **Outputs**: Not explicitly defined, but can include any output from the `configure_supervisor_services`, `reload_supervisor_config`, and `supervisorctl` commands.
- **Returns**: Not explicitly defined, but depends on the success of the `configure_supervisor_services`, `reload_supervisor_config`, and `supervisorctl` commands.
- **Example usage**:

```
hps_services_start
```

This example calls the function without any arguments, resulting in the configuration services being setup, supervisor configuration being reloaded, and all services defined in the supervisor configuration starting up.

2.55.3 Quality and Security Recommendations

- Validate the environment variable `$HPS_SERVICE_CONFIG_DIR` to ensure it contains the correct path to the supervisord configuration file. Incorrect values or paths might lead to unforeseen errors.
- Ensure robust error handling mechanisms are in place to manage failures from `configure_supervisor_services`, `reload_supervisor_config`, and `supervisorctl` commands.
- Confirm that appropriate access controls are enforced around the function usage to maintain security and restrict unauthorized individuals.
- Implement proper logging mechanisms to capture any errors or debug information for future troubleshooting and audits.
- Ensure supervisord services are configured with least required permissions to mitigate any security vulnerabilities.
- Regularly update and patch supervisor and its components to secure against any known vulnerabilities.

2.56 hps_services_stop

Contained in `lib/functions.d/system-functions.sh`

2.56.1 Function overview

The `hps_services_stop` function is a bash function that stops all services managed by the HPS (High Performance Systems) service manager. The function uses the `supervisorctl` command, with a configuration file defined by `HPS_SERVICE_CONFIG_DIR` global variable.

2.56.2 Technical description

- **Name**: `hps_services_stop`
- **Description**: Stops all services managed by the HPS service manager.
- **Globals**:
  - `HPS_SERVICE_CONFIG_DIR`: Absolute path to the directory containing supervisord configuration file.
- **Arguments**: None.
- **Outputs**: Any output or error message from the `supervisorctl` command.
- **Returns**: The exit status of the last command executed, in this case the `supervisorctl` command.

- **Example usage**:

```
# Given the configuration file is at "/etc/hps/supervisord.conf"
export HPS_SERVICE_CONFIG_DIR=/etc/hps
hps_services_stop
```

2.56.3 Quality and security recommendations

- The function should validate that `HPS_SERVICE_CONFIG_DIR` variable is set and is not an empty string.
- It should check if the `supervisord.conf` file actually exists at the specified location before attempting to perform any operations.
- To ensure security, the user running this script should have the minimal privileges necessary to stop the services.
- The function can be enhanced to handle errors and exceptions thrown by the `supervisorctl` command.
- Any outputs and error messages should be logged for later reviewing.

2.57 `initialise_cluster`

Contained in `lib/functions.d/cluster-functions.sh`

2.57.1 Function Overview

The `initialise_cluster` function is utilized for initializing a new cluster in a High Performance System (HPS). It takes as input a cluster name and creates relevant directories and a configuration file within a base directory (default: `/srv/hps-config/clusters`). The function will check if a cluster directory already exists and will stop execution if so. It also uses the `export_dynamic_paths` function to export cluster paths.

2.57.2 Technical Description

- **name**: `initialise_cluster`
- **description**: This function accepts a cluster name and initializes relevant directories and a configuration file within a base directory for the HPS. It also checks if a cluster directory already exists and halts execution if it does. Additionally, it exports cluster paths using the `export_dynamic_paths` function.
- **globals**: `HPS_CLUSTER_CONFIG_BASE_DIR`: The base directory where the cluster directories and files will be stored. Default is `/srv/hps-config/clusters`.
- **arguments**: `[ $1: Cluster name for the new cluster, ]`
- **outputs**: Display whether a new cluster was initialized or if there was an error, and on successful creation, it shows the created config.

- **returns**: 1 if a cluster name was not provided, 2 if the cluster directory already exists, 3 if an error occurred in exporting cluster paths using the `export_dynamic_paths` function.
- **example usage**:

```
initialise_cluster "new_cluster"
```

2.57.3 Quality and Security Recommendations

1. Enhance error handling: Include more comprehensive error handling and reporting. This function currently presents potential avenues for failures, such as if the base directory cannot be written to or if `mkdir` or `cat` fail.
2. Sanitize inputs: Validate the cluster name for potential security issues. For instance, an unexpected input could navigate out of the desired directory structure and overwrite arbitrary files.
3. Consider concurrency: If multiple scripts might run this concurrently, add checks or implement file locking mechanisms to avoid conflicts.
4. Remove global variables: Global variables can make a script harder to reason about and more prone to errors. Consider alternatives such as passing variables as arguments to functions.
5. Robustness: The function should check whether the `export_dynamic_paths` function exists before attempting to call it.

2.58 `initialise_distro_string`

Contained in `lib/functions.d/configure-distro.sh`

2.58.1 Function overview

The function `initialise_distro_string()` is primarily intended to identify and provide a unique string representation of the system's operating distribution (distro), its CPU architecture, version number and manufacturer. The function first fetches the CPU type using the `uname -m` command and the manufacturer is predefined as "linux". Then, it checks for the existence of '/etc/os-release' file. If this file exists, it's sourced and the distro name(`osname`) and version(`osver`) are extracted from the environment variables `ID` and `VERSION_ID` respectively, then converted to lowercase. If the file doesn't exist, 'unknown' is assigned to both `osname` and `osver`. Finally, it concatenates and echoes these information in a specific format - `<cpu>-<mfr>-<osname>-<osver>`.

2.58.2 Technical description

- **Name**: `initialise_distro_string`

- **Description**: This function collects and reports system information in a string format, featuring the architecture, manufacturer, distro name, and version.
- **Globals:**
  - None used explicitly
- **Arguments**: None
- **Outputs:**
  - Echoes a string consisting of CPU architecture, manufacturer, distribution name, and distribution version all separated by hyphen(-).
- **Returns:**
  - None
- **Example usage**:

```
$ initialise_distro_string
```

2.58.3 Quality and security recommendations

1. Add documentation for this function, describing what it does, its inputs, outputs and return values.
2. Implement error handling for if `uname -m` command fails to execute. Currently, it silently fails in that case, which may lead to unexpected results.
3. Check if the variables `ID` and `VERSION_ID` exist in the `/etc/os-release` file before assigning them to `osname` and `osver`.
4. Validate/verify the data read from `/etc/os-release` file. It's a potential security risk to trust this data implicitly.
5. Consider using a more granular access control method for determining whether `/etc/os-release` exists and can be read. Currently, a failure to access this file would default the OS name and version to 'unknown', which may not be a accurate reflection of the system status.

2.59 `initialise_host_scripts`

Contained in `lib/functions.d/configure-distro.sh`

2.59.1 Function overview

The `initialise_host_scripts` function is part of a broader bash script in a system deployment context. This function interacts with a remote server to retrieve a set of functions. Its main actions include initializing the host and fetching the provisioned bundle of bash functions from a remote server, which gets executed locally. If the function encounters any problem during fetching, it notifies the user.

2.59.2 Technical description

- **Name**: initialise_host_scripts
- **Description**: This function aims to automatically initialize the host by fetching a set of shell scripts from a remote server, where these scripts contain bash functions. Once retrieved, these scripts are sourced into the currently running shell.
- **Globals**: None
- **Arguments**: None
- **Outputs**: This function outputs several messages in stdout or stderr about the progress of fetch operation.
- **Returns**: 0 when fetch is successful and the scripts were sourced correctly, 1 when the fetch operation fails.
- **Example Usage**: `initialise_host_scripts`

2.59.3 Quality and security recommendations

- To enhance security, consider using HTTPS instead of HTTP in the URL to fetch the script. HTTPS ensures the encryption of the data during transmission.
- User input validation and error handling could be enhanced to prevent the function from failing silently in case of unexpected input or behavior.
- Sourcing a script from a remote location can be a huge security risk. Malicious code could potentially be included and executed. If not strictly necessary, consider alternative options.
- Use a more robust approach for string concatenation and variable expansion to avoid an unexpected result, such as array syntax.
- Redirecting output errors to a log file will simplify the process of identifying and solving any future problems.

2.60 `int_to_ip`

Contained in `lib/functions.d/network-functions.sh`

2.60.1 Function Overview

The `int_to_ip()` function is designed to convert an integer into IP format. This function is often used to work with IP addresses in scripts, supporting tasks like network conversions and other networking-related operations.

2.60.2 Technical Description

- **Name**: `int_to_ip()`
- **Description**: This function takes an integer as an input and converts it into a corresponding IP address.

- **Globals**: None
- **Arguments**:
  - `$1`: The integer value that needs to be converted into an IP address.
- **Outputs**: Outputs the IP address corresponding to the input integer.
- **Returns**: None
- **Example Usage**: `bash local network="192.168.1.0" local broadcast="192.168.1.255" local gateway_ip="192.168.1.1" local net_int=$(ip_to_int "$network") local bc_int=$(ip_to_int "$broadcast") local gw_int=$(ip_to_int "$gateway_ip") echo "$(int_to_ip "$gw_int")"`

2.60.3 Quality and Security Recommendations

1. The function suffers from a lack of input validation. Any integer can potentially be passed into the function. Therefore, ensure to validate the integer before conversion to avoid any potential errors or security vulnerabilities.
2. You should add error handling in case of unexpected input (e.g., non-integer values) or unexpected return values.
3. Consider enclosing the whole block of code into another function with its specific functionality. It would increase the reusability of your code.
4. Add comments to improve code readability and maintainability.
5. Consider returning meaningful error messages instead of letting the script fail silently.
6. Security-wise, sanitize all inputs to the bash function in order to prevent injection attacks.
7. Update your function to use the standard error output (stderr) for error messages.

2.61 `ip_to_int`

Contained in `lib/functions.d/network-functions.sh`

2.61.1 Function Overview

The **ip_to_int()** function is used to transform an IPV4 address that was passed as an argument, into an integer equivalent. It does this by splitting the IP address into its four octets and executing bit-shift operations followed by additions.

2.61.2 Technical Description

- **Name:** ip_to_int()
- **Description:** This function converts an IPV4 address, passed as a string, into an equivalent integer. The IP address is split into octets which undergo bit-shift

operations and are then added together for the result.

- **Globals:** None
- **Arguments:**
  - **$1:** IP address string to be converted.
- **Outputs:** An integer corresponding to the input IPV4 address.
- **Returns:**
  - Returns the integer equivalent of the input IPV4 address.
- **Example usage:**
  - `ip_to_int "192.168.1.1"`
  - The outputs will be `3232235777` which stands for the integer equivalent of IP address “192.168.1.1”.

2.61.3 Quality and Security Recommendations

1. Input validation is always a good practice. Ensure that the input string meets all criteria for a valid IPV4 address before proceeding with conversion.
2. Use local variables instead of globals whenever possible, as globals can be manipulated in other parts of the script, potentially leading to unexpected results.
3. Always double-quote your variables when using them. This prevents word splitting and pathname expansion.
4. Error handling should be added for cases where there are input or computation errors for improved reliability.

2.62 `ipxe_boot_from_disk`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

2.62.1 Function Overview

The function `ipxe_boot_from_disk` is designed to boot from the local disk via BIOS by exiting the iPXE stack. It first calls another function `ipxe_header` to prepare the iPXE environment. Then it issues a series of `echo` commands to instruct the system to print a message, delay for some time, and eventually exit the iPXE environment, handing over control to BIOS for booting from the local disk.

2.62.2 Technical Description

- **Name**: ipxe_boot_from_disk
- **Description**: This function is used to boot from the local disk via BIOS by exiting the iPXE stack. It first initiates the iPXE environment with `ipxe_header`, then prints a message, adds a small pause, and exits the system.
- **Globals**: None

- **Arguments**: No arguments are accepted by this function.
- **Outputs**: The function prints out the message “Handing back to BIOS to boot”
- **Returns**: Does not return a value.
- **Example Usage**:

```
ipxe_boot_from_disk
```

2.62.3 Quality and Security Recommendations

1. Implement a function guard to prevent the function from being sourced and run unintentionally.
2. Operate robustness in edge cases, specifically, specify what happens if `ipxe_header` function doesn’t execute as expected.
3. Add error handling for external calls (in this case `ipxe_header`), ensuring the function can handle if any error occurs.
4. Considering the `echo` command could potentially be hijacked for arbitrary command execution, use the `printf` function for safer output.
5. Because the function does printf to stdout directly, consider adding an option for silent running.
6. Always consider the implications of allowing your script to sleep in a production environment. Could this cause other operations to time out? Could it allow race conditions to introduce errors?
7. Make sure that the function does not contain hardcoded secrets or sensitive information.
8. Check for the necessary permissions before attempting to boot from disk, gracefully handle the case if permission is denied.
9. A potential improvement could be to make the sleep time configurable as different systems may require different wait times before booting.

10. A security improvement could be to verify that booting from BIOS was successful and no errors were thrown during the process.

2.63 `ipxe_boot_installer`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

2.63.1 Function overview

The `ipxe_boot_installer` function in Bash is primarily used to boot a system over the network using iPXE (an open source boot firmware). It takes a host type and a profile as arguments and configures the host for booting. The function is highly versatile with a number of steps including loading host type profiles, getting host type parameters, checking the state of the host, setting the host type and profile, mounting the distribution

ISO, checking if the kernel file exists, and preparing the iPXE boot for the operating system installation.

2.63.2 Technical description

Below is a pandoc definition block of the `ipxe_boot_installer` function :

- **name**: ipxe_boot_installer
- **description**: A function for configuring a host for booting over the network using iPXE with a specific host type and profile.
- **globals**: [HPS_DISTROS_DIR: a directory path storing distros, CGI_URL: endpoint for kickstart file, mac: address for the host machine, CPU,MFR,OSNAME, OSVER: parameters for host, KERNEL_FILE,INITRD_FILE: whereabouts of kernel image and initrd image in distro respectively.]
- **arguments**:
  1. $1: host_type (the type of host to be configured)
  2. $2: profile (the profile to be used for the configuration)
- **outputs**: Console outputs related to booting log, error messages.
- **returns**: An iPXE boot install sequence for the respective OS.
- **example usage**: `ipxe_boot_installer server Linux`

2.63.3 Quality and security recommendations

1. Be sure to sanitize and validate inputs. This function seems to trust the `host_type` and `profile` parameters without verifying their integrity.
2. Add comments to code snippets that might be hard to understand, providing an insight of what the piece of code is actually doing.
3. Error handling should be improved. Adding more specific error messages can help pinpoint the issue causing the error.
4. Whenever possible, use absolute paths for file or directory references to make sure they are being correctly located.
5. The function deploys a set of commands in EOF which presents potential security issues. Avoid using EOF blocks to construct command sequences where feasible.

2.64 `ipxe_cgi_fail`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

2.64.1 1. Function Overview

The function `ipxe_cgi_fail` is used in error handling within the iPXE boot firmware configuration. This function is typically called when there is a failure or an error occurred during the network booting process. When called, it displays an error message, after which it awaits for 10 seconds before initiating a system reboot.

2.64.2 2. Technical Description

The below description in detail explains each component of the `ipxe_cgi_fail` function.

- **Name:** `ipxe_cgi_fail`
- **Description:** This function is invoked when an error occurs during the iPXE boot process. It generates an iPXE header using the `ipxe_header` function, logs the error message using the `hps_log` function, and then prints the error message on the screen. Finally, it puts the system into sleep for 10 seconds and performs a reboot.
- **Globals:** None.
- **Arguments:**
  - `$1: cfmsg` - This represents the custom fail message, which is passed as an argument to the function.
- **Outputs:** This function logs the error message format to the console: "== ERROR ==". Also, it sets the iPXE to sleep for 10 seconds and then reboot.
- **Returns:** This function does not return any value because once it is invoked, it exits the current execution and initiates a reboot after a sleep duration of 10 seconds.
- **Example Usage:**

```
ipxe_cgi_fail "Network boot failed"
```

2.64.3 3. Quality and Security Recommendations

Here are some quality and security improvements recommended for the `ipxe_cgi_fail` function:

- Logging: Improve logging to include information about the specific error and the environment that caused the error.
- Validation: Validate the input parameter to prevent potential code injections or crashes due to undefined variables.
- Error Handling: A defined strategy for error handling to guarantee the system stability and high-availability performance after the execution of this function.
- Communication of Errors: It might make sense to communicate major errors to a central error log or even an alerting system, so they do not go unnoticed.
- Sleep & Reboot strategy: Depending on your specific situation, you might need to review your system's sleep and reboot strategy for better performance or for minimizing downtime.

2.65 `ipxe_configure_main_menu`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

2.65.1 Function overview

The function `ipxe_configure_main_menu` is used to generate the main interaction menu for host configuration in a cluster network via iPXE (Open Source Boot Firmware). It is responsible for presenting a list of possible actions to the user if they have a configured cluster but their host is not adequately configured yet. Depending on the configuration state (specifically the `FORCE_INSTALL` state), it may provide options to force installation and wipe disks, among other features.

2.65.2 Technical description

- **Name**: ipxe_configure_main_menu.
- **Description**: This function dynamically generates a main menu system for hosts not yet configured in a cluster network. It populates the menu based on the state of `FORCE_INSTALL`. It also creates an infrastructure for logging and handling the user's menu selection.
- **Globals**: [FORCE_INSTALL_VALUE: Holds the state of the `FORCE_INSTALL` configuration parameter, mac: holds the mac address].
- **Arguments**: This function does not take any arguments.
- **Outputs**:
  - The menu layout and the set of options available, which is echoed to the standard output.
  - The chosen selection is fetched and the process of that selected menu item is initiated.
- **Returns**: Nothing is explicitly returned.
- **Example usage**: `ipxe_configure_main_menu`

2.65.3 Quality and security recommendations

1. Prevent code injection by ensuring the integrity of the `FORCE_INSTALL_VALUE`.
2. The function uses `cat <<EOF`, which can potentially expose sensitive data. Always ensure the content within this block is sanitized.
3. To improve code readability, consider adding more comments explaining the logic behind certain actions, such as the way `FORCE_INSTALL_VALUE` is handled.
4. Ensure exception handling during operations like `imgfetch` to maintain robustness.
5. Avoid logging sensitive information to prevent potential leaks.

2.66 `ipxe_header`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

2.66.1 Function Overview

This bash function, `ipxe_header`, is designed to create and send a Preboot Execution Environment (PXE) header, preventing boot failure. The function also sets some variables to be used in IPXE scripts. It manifests an IPXE script with the log message, image fetch command, and several echo statements for successful connection to a cluster, client IP, and MAC address.

2.66.2 Technical Description

Name: `ipxe_header`

Description: The function, `ipxe_header`, is used to send a Preboot Execution Environment (PXE) header and to define variables for IPXE scripts. The function provides user feedback about the connection state and prints out the client's IP along with its MAC address.

Globals: - `CGI_URL`: used as the URL for the image fetch in IPXE script. - `TITLE_PREFIX`: defines the prefix for any titles in the IPXE script.

Arguments: None

Outputs: - IPXE script with log message, image fetch, and echo statements regarding the connection to the cluster, and the client's IP and MAC address.

Returns: None.

Example of usage: To use the function, simply call it from your bash script:

```
ipxe_header
```

2.66.3 Quality and Security Recommendations

- To improve the quality, consider adding error handling mechanisms, for instance, in cases where the `cgi_header_plain` function fails.
- Be mindful of shellcheck warnings. This tool helps in identifying and fixing potential bugs in the script.
- The current function does not take any arguments. However, if the function's complexity increases in future requiring it to accept user inputs, sanitize any user inputs to prevent possible Injection attacks.
- Avoid clear-text transmission of sensitive data and opt for HTTPS instead of HTTP for the `CGI_URL` if feasible.
- Consider implementing a logging mechanism that would log any potential runtime errors. This may prove useful for troubleshooting.

2.67 ipxe_host_install_menu

Contained in `lib/functions.d/ipxe-menu-functions.sh`

2.67.1 Function overview

The `ipxe_host_install_menu` function generates an interactive installation menu for an iPXE host. The displayed menu provides the user with four different installation options: a Thin Compute Host, a Storage Cluster Host, a Disaster Recovery Host, or a Container Cluster Host. The selected option is logged and then processed.

2.67.2 Technical description

name: `ipxe_host_install_menu`

description: This function is meant to be called without parameters. It generates a series of option prompts for a user to interactively choose an installation type for an iPXE host using combustion menus. The user's selection is logged and sent to a processing command.

globals: - `TITLE_PREFIX`: Description (initially undefined) - `CGI_URL`: Description (initially undefined)

arguments: - None

outputs: The function outputs a menu with different installation options for an iPXE host.

returns: The function does not explicitly return a value but the output of the user's selection is passed to a processing command.

example usage:

```
ipxe_host_install_menu
```

2.67.3 Quality and security recommendations

1. To improve security, consider validating the value of `${mac:hexraw}` and `${selection}` variables before using them in the command to prevent possible command injection.
2. User inputs should be handled carefully. Ensure that the user can only choose pre-specified options and cannot input arbitrary values.
3. Include comments within your function to better explain the purpose and functionality of different parts of your code.
4. Consider error handling for the case when the user doesn't select any menu item.
5. It would be more maintainable to have a dynamic way to add or remove installation options.
6. Consider making sure that `ipxe_header` function (that is being called) exists and works as expected.
7. You may want to check and verify any external dependencies such as `imgfetch`, `chain`, and `CGI_URL`.

2.68 ipxe_host_install_sch

Contained in lib/functions.d/ipxe-menu-functions.sh

2.68.1 Function Overview

ipxe_host_install_sch is a function within the Bash environment that displays a menu to install a Storage Cluster Host (SCH). The function implements the iPXE protocol for boot-loading and includes options for single-disk ZFS installation (for testing or multiple hosts) as well as ZFS RAID installations (for multiple local disks).

2.68.2 Technical Description

- **name**: ipxe_host_install_sch
- **description**: This Bash function displays an installation menu for a Storage Cluster Host (SCH) using the iPXE protocol. It provides options for both single-disk ZFS and ZFS RAID installation types.
- **globals**: *N/A*
- **arguments**: *N/A*
- **outputs**: This function will output a text-based installation menu to the user's terminal environment.
- **returns**: The function does not return a value. It executes code to display the installation menu and execute the installation immediately based on the user's selection.
- **example usage**: Run the function without any parameters in a bash shell like so: ipxe_host_install_sch.

2.68.3 Quality and Security Recommendations

1. Implement error checking and handling to ensure that the function behaves as expected.
2. Consider including usage details or help instructions in the menu to guide users.
3. To enhance security, the communication between local and network environments should be encrypted. This includes information relating to the installation action and logging messages.
4. Add validation checks for the menu selections to avoid potential command injection vulnerabilities.
5. Always use the latest version of iPXE protocol for better support and security features.
6. Consider adding user confirmation before executing the installation to avoid potential data loss or overwriting.

2.69 ipxe_init

Contained in lib/functions.d/ipxe-menu-functions.sh

2.69.1 Function overview

The ipxe_init function is a process involved within a cluster. This function comes into play when the cluster is configured and the host is not yet identified due to the absence of the MAC address. It provides an iPXE menu to unknown hosts. It seeks to find a configuration and load it based on the MAC address and a CGI_URL. This function prints out the config URL, fetches the config, loads it, states the image status, and executes the config. There is a commented section in place to handle situations where no host configuration is found.

2.69.2 Technical description

- **name**: ipxe_init
- **description**: This function deals with the IPXE initialization process for unidentified hosts. It creates a request using the host's MAC address and a CGI_URL. After fetching and loading the configuration for the host, it gives the image status and then executes the configuration.
- **globals**:
  - CGI_URL: This is a predefined URL used for making HTTP requests.
- **arguments**: none
- **outputs**:
  - The URL from where the configuration has been requested.
  - The status of the loaded image.
- **returns**: none
- **example usage**: ipxe_init

```
ipxe_init
```

2.69.3 Quality and security recommendations

1. Adding error handling apart from the commented section
2. Ensuring that the CGI_URL is defined and valid
3. Checking that the host MAC address used in the config_url is valid
4. Securing the CGI_URL as it may expose sensitive information
5. There are commented lines of codes in the function. For code readability, remove them if they are not necessary or provide adequate explanations otherwise.

2.70 ipxe_reboot

Contained in lib/functions.d/ipxe-menu-functions.sh

2.70.1 Function overview

The `ipxe_reboot` is a Bash function that initiates a reboot procedure and logs this activity. The function accepts a custom message as a parameter, which influences the activities during the reboot process. The function first logs the call to reboot, provides an iPXE header, then depending on presence or absence of the custom message, it echoes that message or merely proceeds to announce a reboot initiation. Then, the system suspends the operation for 5 seconds, and finally reboots.

2.70.2 Technical description

```
name: ipxe_reboot
description: A bash function used to initiate and log a reboot
 ↪  activity on a system.
globals: [ MSG: the custom message that is echoed during the reboot
 ↪  process ]
arguments: [ $1 (MSG): Custom message to be logged and echoed
 ↪  during the reboot sequence ]
outputs:
   - Log messages about the reboot.
   - Echoed messages either custom MSG or standard "Rebooting..."
 ↪  message.
returns: The function itself does not have any return command,
 ↪  hence nothing gets returned.
example usage:
- ipxe_reboot "System update performed, rebooting now."
- ipxe_reboot "Unexpected error occurred, rebooting."
```

2.70.3 Quality and security recommendations

1. Always ensure to sanitize and validate the custom message (MSG) input to prevent command injection attacks.
2. Add an optional timeout parameter to give users control over sleep duration instead of hardcoding sleep as 5 seconds. This might be useful in different operational conditions.
3. The logging functionality (`hps_log`) should have proper permission checks to avoid unauthorized access and potential data tampering. Additionally, it should handle log management aspects such as log rotation, size limitations and sensitive data exclusion.
4. If possible, avoid using direct console output (via `echo`) for important messages. Consider using a logging mechanism instead.
5. The function does not check whether the reboot command succeeds or not. It could be improved by adding error handling to catch any failure during the reboot process and respond accordingly.

2.71 ipxe_show_info

Contained in lib/functions.d/ipxe-menu-functions.sh

2.71.1 Function overview

The ipxe_show_info function is designed for displaying various pieces of information about the host system, its configuration and the iPXE boot environment. The function uses a switch-case structure to organise the printing of information into distinct sections such as iPXE info, cluster configuration, host configuration and system paths. All of this information can be valuable for system troubleshooting and administration purposes.

2.71.2 Technical description

- Name: ipxe_show_info
- Description: This function is used to display information about the iPXE environment, the host and the cluster. The information to be displayed is determined by the category argument passed to the function.
- Globals: [HPS_CLUSTER_CONFIG_DIR: The directory where the cluster configuration is stored, HPS_CONFIG: The location of the HPS configuration file]
- Arguments: [$1: The category of information to be displayed. Can be one of the following: show_ipxe, show_cluster, show_host, or show_paths]
- Outputs: Outputs information in the terminal about the host, iPXE environment, cluster and system paths, based on the chosen category.
- Returns: Does not return a value.
- Example usage: ipxe_show_info show_cluster

2.71.3 Quality and security recommendations

- Consider adding input validation for the category argument to ensure that only valid categories are processed.
- When displaying file content, make sure to handle any potential errors (file not found, permission denied, etc.) in a way that maintains the function's stability and security.
- Avoid disclosing sensitive info (passwords, API keys etc.) within the displayed information. You should regularly review the displayed info to ensure no confidential data is unintentionally exposed.
- The function makes use of several global variables. To improve encapsulation, consider altering the function to accept these values as arguments instead.

2.72 list_clusters

Contained in lib/functions.d/cluster-functions.sh

2.72.1 Function Overview

The `list_clusters` function essentially lists all the clusters available in the High Performance Storage (HPS) cluster base directory. It accomplishes this by using a Glob function to capture the file path of all directories within the base directory. It then conveniently removes the base file path, leaving only the name of the cluster directories.

2.72.2 Technical Description

- **Name**: list_clusters
- **Description**: This command operates in the bash shell to enumerate all directories (representing clusters) present within a specified base directory. It uses the globbing option of the bash shell to access the file paths to all directories in the base directory. The base file path is removed from each resulting string leaving only the name of the cluster which is then printed to the console.
- **Globals**: [HPS_CLUSTER_CONFIG_BASE_DIR: the base directory which contains directories representing clusters]
- **Arguments**: [This function does not take arguments]
- **Outputs**: Prints the names of the cluster directories to the console
- **Returns**: NULL
- **Example usage**:

```
list_clusters
```

2.72.3 Quality and Security Recommendations

To increase quality and security some recommendations are as follows:

1. Validate the existence of `HPS_CLUSTER_CONFIG_BASE_DIR` before using it. If not set, the function should handle the error gracefully and notify the user.
2. Ensure only authorized users can execute this function. Information about directories might be sensitive.
3. Ensure that all file reads from the directories are secure and sanitize any user-provided input to prevent directory traversal or other filesystem attacks.
4. Validate that the found directories match expected patterns to avoid unexpected behavior or security issues.
5. Handle edge case when there are no directories / clusters.
6. Add comments in the function to increase maintainability.

2.73 `list_local_iso`

Contained in `lib/functions.d/iso-functions.sh`

2.73.1 Function overview

The bash function `list_local_iso()` is designed to search for local ISO files in a specified directory based on the provided parameters (cpu, manufacturer, operating system name and its version). If the optional `osver` argument is supplied, then it is included in the search pattern. The function lists all the found ISO files with corresponding names. If no matches are found, the function returns an echo statement indicating that no matching ISO files were found.

2.73.2 Technical description

- **name**: list_local_iso
- **description**: Searches for ISO files in the ISO directory specified by the `get_iso_path` function, based on the `cpu`, `mfr`, `osname` and `osver` parameters. If the `osver` parameter is not supplied, then it is left out of the search pattern.
- **globals**: N/A
- **arguments**: [$1: cpu, $2: mfr, $3: osname, $4 (optional): osver]
- **outputs**: A list of found ISO files or a message indicating that no ISO files were found.
- **returns**: 1, if no ISO files were found.
- **example usage**:

```
list_local_iso 'x86' 'intel' 'ubuntu' '18.04'
```

2.73.3 Quality and security recommendations

1. Validate the input parameters.
2. Add error handlers for potential issues - for example, what happens if the directory specified by `get_iso_path` does not exist.
3. Make sure that `get_iso_path` provides a correct path to prevent potential path traversal attacks.
4. Output all echo messages not only to the standard output but also to a dedicated log file with timestamps for better tracking.
5. Make sure the function works as expected with special characters in the `cpu`, `mfr`, `osname`, and `osver` parameters.
6. Implement an exit mechanism to break the loop if it runs for a certain amount of time to avoid potential infinite looping.

2.74 load_cluster_host_type_profiles

Contained in `lib/functions.d/cluster-functions.sh`

2.74.1 Function overview

The function `load_cluster_host_type_profiles` loads the configurations of a cluster from a given file. The function reads the active cluster filename and checks if the file exists before proceeding. It then declares a global associate array for storing the host types. The function reads the file line by line, checks for key-value pairs, and stores these pairs in the associative array.

2.74.2 Technical description

Name: load_cluster_host_type_profiles

Description: This function is designed to load the profiles of host types in a cluster setting. It reads from a configuration file and populates an associative array with the host types and their corresponding key-value pair profiles.

Globals: [__declared_types: This associative array stores the types of hosts in the cluster.]

Arguments: [$1: The first argument represents the key in the key-value pair, $2: The second argument represents the value in the key-value pair.]

Outputs: The function does not print any output, instead, it populates a global associative array.

Returns: The function will return 1 if the configuration file does not exist; otherwise, no explicit return value is given, which implies a default return value of 0 indicating a successful execution.

Example Usage:

```
load_cluster_host_type_profiles
```

No parameters are needed to call this function. The specific configuration file and details are handled within the function itself.

2.74.3 Quality and Security recommendations

1. Add error handling for file reading operations to catch and handle errors during the execution.
2. Perform additional validation on each key-value pair read from the configuration file to ensure they are in the expected format.
3. Provide clear documentation regarding the structure and content of the configuration file to avoid potential misconfiguration by users.
4. Keep the function updated per any changes to the configuration file structure or content for consistency, avoiding potential issues and errors.

2.75 make_timestamp

Contained in lib/functions.d/system-functions.sh

2.75.1 Function Overview

The Bash function, make_timestamp, returns the current date and time in the format Year-Month-Day Hours:Minutes:Seconds UTC. The time is represented in Coordinated Universal Time (UTC), providing a standardized timestamp.

2.75.2 Technical Description

Name:

- make_timestamp() ##### Description:
- The function uses the date command with the -u option, which represents the current date and time in UTC. The format is specified as +'%Y-%m-%d %H:%M:%S UTC', which formats the date as Year-Month-Day and the time as Hours:Minutes:Seconds, followed by the string 'UTC'. ##### Globals:
- Not applicable. ##### Arguments:
- The function does not accept any arguments. ##### Outputs:
- The function outputs the current date and time in UTC, in the format 'YYYY-MM-DD HH:MM:SS UTC'. ##### Returns:
- The function does not return any values, it only has an output. ##### Example Usage:

```
echo "$(make_timestamp)"
```

In the console, it may appear as:

```
2022-06-14 12:07:28 UTC
```

2.75.3 Quality and Security Recommendations

- As the function is quite simple, there isn't much room for quality improvements. However, it is always good practice to comment your code for better readability.
- From a security perspective, the date command is safe to use. It does not take user input or access sensitive data. However, if your application has specific security requirements around time handling or time zones, you may need to consider these.
- If you want to use this function in a larger script, you might want to consider error handling, even if the chances of the date command failing are exceptionally low.

2.76 mount_distro_iso

Contained in lib/functions.d/iso-functions.sh

2.76.1 Function Overview

`mount_distro_iso` is a Bash function that is used for mounting ISO images of different Linux distributions on your system. It takes two arguments: `DISTRO_STRING` which is the string identifier of the Linux distribution, and `iso_path` which is the path to the ISO file. Using these, the function mounts the ISO file to a provided mount point. If the ISO file or the mount point does not exist, it logs error messages and returns appropriate values.

2.76.2 Technical Description

- **Name:** `mount_distro_iso`
- **Description:** This function mounts a given Linux distribution ISO file to a defined mount point. It first checks if the ISO file exists and if the mount point is already in use. If the ISO file doesn't exist, it exits with a return value of 1 indicating an error. If the mount point is already in use, it exits with a return value of 0, indicating that no new action is required. Otherwise, it creates the mount point directory if it doesn't exist, then mounts the ISO file to it.
- **Globals:** `HPS_DISTROS_DIR` describes the directory where the distributions' ISO files are stored.
- **Arguments**:
  - `$1`: `DISTRO_STRING`, a string representing the name of the Linux distribution, used to find the ISO within the specified directory.
  - `$2`: `iso_path`, path where the ISO file of the distribution is located.
- **Outputs:** Information and error logging information directly to stdout.
- **Returns:** Returns 1 if the required ISO isn't found or 0 if no actions are needed as the ISO is already mounted. No return value is mentioned when actions have been successfully performed, which means it reduces to exit status of the last `mount` command.
- **Example usage:** `mount_distro_iso ubuntu ./iso/ubuntu.iso`

2.76.3 Quality and Security Recommendations

1. The function should validate its input arguments to mitigate potential command injection vulnerabilities.
2. It should return a distinct status code in case of success.
3. The error messages should be sent to stderr instead of stdout.
4. Check if the file being mounted is indeed an ISO file with file extension checks or file magic checks.
5. The function should log detailed errors from the `mount` command on failure.
6. Add an unmount feature as well, to return the system to its original state after operations on the mounted ISOs are done to save system resources.
7. Document and handle abnormal behaviors such as lack of permissions to create a directory or to mount files.

2.77 normalise_mac

Contained in lib/functions.d/network-functions.sh

2.77.1 Function Overview

The function normalise_mac normalizes the format of a provided MAC (Media Access Control) address. This is done by removing all common delimiters (periods, hyphens, spaces, colons) and converting all alphabetic characters to lowercase. The function then validates if the MAC address is accurate, meaning it must be exactly 12 hexadecimal characters. If the address is valid, it outputs the normalized MAC address. If not, it sends an error message to stderr and returns 1, signifying an error.

2.77.2 Technical Description

- **Name**: normalise_mac
- **Description**: This function normalizes a MAC address by eliminating delimiters and converting to lowercase, then validates the format.
- **Globals**: None.
- **Arguments**:
  - $1: MAC MAC address to be normalized
- **Outputs**:
  - Normally, the normalized MAC address.
  - In case of error, an error message to stderr.
- **Returns**:
  - 0 if successful.
  - 1 if an error occurs (invalid MAC address format).
- **Example usage**:

```
normalise_mac "00:00:00:aa:bb:cc"
```

2.77.3 Quality and Security Recommendations

1. The function assumes the input is a string, without checking this explicitly. Ensuring this beforehand could prevent errors.
2. Error messages could provide more information to the user about why the format was incorrect (for example, too few/many characters, invalid characters etc.).
3. For further security, consider sanitizing the input to prevent command injection or other potential security risks.
4. Always use "${var}" instead of "$var" to avoid word splitting and globbing problems particularly when dealing with variables that can have spaces. This can result in unexpected behavior.
5. It might be handy to have an option to return the normalized format in different styles, i.e., with certain delimiters, or uppercase.

2.78 prepare_custom_repo_for_distro

Contained in `lib/functions.d/repo-functions.sh`

2.78.1 Function overview

The `prepare_custom_repo_for_distro` function configures a custom repository for a particular Linux distribution. This function accepts a string that identifies the Linux distribution and any number of other parameters that can contain package source information or names of required packages. The function will create a directory for the new repository, download or copy all packages from the provided sources to this directory, and then verify if all required packages are present.

2.78.2 Technical description

Name: prepare_custom_repo_for_distro

Description: The function prepares a custom repository for a specific Linux distribution. It allows to download and locate all necessary packages into the created directory, later verifying that all required packages are present.

Globals: [HPS_PACKAGES_DIR: directory path where packages of various distributions are stored]

Arguments: -[$1 : identifierstringofaLinuxdistro$] — [@: package source links or file paths, and names of necessary packages]

Outputs: - Creates a new directory for a repository if it doesn't exist - Retrieves or copies package files to the new directory - Logs error messages when issues occur

Returns: - Returns 0 if the repository was successfully prepared - Returns 1 if the directory creating process fails - Returns 2 if the downloading process fails - Returns 3 if the file copy process fails - Returns 4 if the package source is invalid - Returns 5 if the metadata creating process fails - Returns 6 if any required packages are missing

Example Usage:

```
prepare_custom_repo_for_distro "ubuntu" "vim"
 ↪  "https://example.com/package.deb"
```

2.78.3 Quality and security recommendations

1. Enforce stricter validation of input. This includes vetting the provided URLs for malicious content.
2. Handle file permission correctly when creating new directories and files.
3. Consider catching errors with specific error handling functions.
4. Use checksums or cryptographic hashes(like SHA256) to ensure the integrity of downloaded packages.

5. Check if there are redundant process which can be eliminated or optimized.
6. Log more detailed information for debugging in case of error.
7. Implement testing to catch and find bugs before production.

2.79 `print_cluster_variables`

Contained in `lib/functions.d/cluster-functions.sh`

2.79.1 Function overview

The Bash function `print_cluster_variables()` is primarily used to read, interpret and display cluster configuration data. The configuration file is obtained from an active cluster by means of another function call. The function reads through the configuration file line by line, where each line is expected to have its keys and values separated by an '='. In the process of reading, it ignores blank lines and comments, while also stripping out the surrounding quotes of the values before displaying them. If the configuration file does not exist, it informs the user and terminates the operation.

2.79.2 Technical description

- **Name**: `print_cluster_variables`
- **Description**: This function reads a configuration file of an active cluster, processes its content to extract the key-value pairs, and prints them. Blank lines and comments are ignored during the process. If the configuration file does not exist, an error message is printed.
- **Globals**: None
- **Arguments**: None
- **Outputs**: Prints the key-value pairs found in the configuration file of the active cluster, or a configuration file not found error message.
- **Returns**: `1` if the configuration file does not exist, otherwise no explicit return value.
- **Example usage** `print_cluster_variables`

2.79.3 Quality and security recommendations

- It's recommended to add error handling for when the `get_active_cluster_filename` function call fails or returns an invalid filename.
- Incorporate permissions validations to ensure only authorized and authenticated users can access this function.
- A key-value pair validation can help mitigate risks of processing malformed data from the configuration file.

- Implementing a secure method of password or sensitive information evasion when printing the variables is recommended to ensure that no sensitive data is exposed in logs or console outputs.
- Additionally, sanitize outputs to avoid any potential command injections or related security threats.

2.80 register_source_file

Contained in lib/functions.d/prepare-external-deps.sh

2.80.1 Function overview

The function register_source_file is essentially a source file registration function in Bash. It takes a filename and a handler as arguments and registers them as a source file in a specific directory (HPS_PACKAGES_DIR, which defaults to /srv/hps-resources/packages/src). It avoids duplicate entries by checking their existence before registration. If the source file already exists it will echo a message stating the file is already registered. If not, it will register the file and handler in the index_file.

2.80.2 Technical description

- **name**: register_source_file
- **description**: A Bash function that checks and registers a filename and handler as a source file in an index file found in the HPS_PACKAGES_DIR directory, or the default directory if not set. It prevents registering duplicate entries.
- **globals**: [HPS_PACKAGES_DIR: A variable for the target directory where the source files are registered. If not set, the default directory is /srv/hps-resources/packages/src.]
- **arguments**: [$1: Filename to be registered,$2: Handler for the file]
- **outputs**: Echoes the action (registration or lack thereof due to duplicates) and the file and handler details.
- **returns**: 0 (when the file is already registered)
- **example usage**: register_source_file "newfile.sh" "myhandler"

2.80.3 Quality and security recommendations

1. Consider adding checks for the validation of arguments provided. For instance, ensuring that they are not null or verifying their format improves robustness.
2. Include error handling for the cases when the target directory cannot be created. This will be helpful for diagnosing potential issues.
3. Using absolute path names can enhance the script's security to avoid any potential relative-path attacks.

4. Inputs should be sanitized to prevent possible injection attacks. For example, if the handler is not securely sanitized, an attacker might manipulate it to execute arbitrary code.
5. Always use the double quotes around variable interpolations to prevent word splitting and glob expansion.
6. Implement clear and detailed logging. In addition to the operation result, these logs could include timestamps, user IDs, and more granular details about the operations. This improves traceability and debugging in case of unexpected behavior or errors.

2.81 `reload_supervisor_config`

Contained in `lib/functions.d/configure-supervisor.sh`

2.81.1 Function Overview

The `reload_supervisor_config` function is specifically designed for managing changes made to the multiple processes or programs run by the supervisor. It allows the supervisor to reread its configuration file and then make an update for changes to take effect. The function requires certain global variables for it to work and two primary actions are taken in its execution; rereading the configuration file and updating the supervisor with the new configurations.

2.81.2 Technical Description

Name: reload_supervisor_config

Description: This function is used to reload the configuration file of supervisord. It first reads the configuration file again and then updates it.

Globals:

- SUPERVISORD_CONF: This variable stores the path to the configuration file for supervisord.

Arguments:

- HPS_SERVICE_CONFIG_DIR: This is where the `supervisord.conf` file is located.

Outputs: There is no specific output as this function simply executes commands.

Returns: There is also no specific return value as the function executes commands directly and their success or failure would be reflected in their respective side effects.

Example Usage:

```
HPS_SERVICE_CONFIG_DIR="/etc/supervisor" reload_supervisor_config
```

2.81.3 Quality and Security Recommendations

1. Always ensure that the correct permissions are granted for the supervisord configuration file ((usually `supervisord.conf`). It should be readable by the application, but not writable to avoid accidental or malicious changes.
2. Consider adding error handling in the event that the configuration file is not found or is not parseable. This could be done by checking the return status of the `supervisorctl` commands ($?).
3. Consider enhancing this function by adding echo statements that display informative messages before and after updating the configurations so users can easily follow along.
4. It is a good practice to use absolute file paths instead of relative ones to prevent potential issues with file locations.
5. To ensure the function doesn't unintentionally modify the global variable, consider passing the configuration directory path as a function argument instead of implicitly relying on the global variable HPS_SERVICE_CONFIG_DIR.

2.82 remote_function_lib

Contained in `lib/functions.d/kickstart-functions.sh`

2.82.1 Function Overview

The `remote_function_lib` function is used as a container for other functions that are to be injected in pre and post sections of a script. It outputs the functions as Here Documents (EOF) which is a technique used to output a multiline string, thereby avoiding any lexical scoping issues that may arise during execution. These functions could be used across scripts, hence, the advantage of centralizing them with this function.

2.82.2 Technical Description

Name: `remote_function_lib`

Description: This function acts as a library for other functions which are to be used in scripts' pre and post sections. It injects these functions by using the "cat" command along with EOF to hold these functions as heredoc strings. Currently, the function does not have any additional custom functions inside but could be populated accordingly.

Globals: None

Arguments: The function does not take any arguments

Outputs: Prints the functions that are to be injected in pre and post sections via STDOUT.

Returns: None

Example Usage: Depending on the functions needed in the pre and post sections of the script, they could be added into the `remote_function_lib` function. Below is a hypothetical usage:

```
remote_function_lib () {
cat <<EOF
sample_function () {
  echo "This is a sample function"
}
EOF
}
```

2.82.3 Quality and Security Recommendations

- To improve reusability and maintainability, consider moving this function into separate standalone file, especially if the function lists are growing larger.
- Add a parameter to the function that specifies which functions to include. This way, you can have one large library of functions, and only include the ones you need.
- Ensure that the inserted code is free of malicious content or bugs. This requires validation, either manually or through automated tests.
- Consider making this function read-only to prevent any unauthorized changes. In bash, you can do this using the `readonly` keyword.
- Always check and handle errors. Since you're using `cat` to insert functions, you should check its return value to make sure it succeeded. If not, stop execution and print an error message.

2.83 `remote_log`

Contained in `lib/functions.d/kickstart-functions.sh`

2.83.1 Function overview

The `remote_log()` function is designed for sending log messages to a remote server. The function accepts a message as input, then URL-encodes the message, and finally posts the encoded message to a predetermined gateway using the curl command in a bash environment. This function could be useful in scenarios where log data is gathered on one machine but needs to be sent and logged to a remote server for centralized logging and analysis.

2.83.2 Technical description

Here is the detailed definition block for the `remote_log` function:

- **name**: `remote_log`.
- **description**: This function encodes an input message in URL format and sends it to a specified gateway.
- **globals**: [`macid`: Used in the URL for the log message to the remote server. Should contain the MAC address of the machine, `HOST_GATEWAY`: Gateway server where the encoded log message is sent.]
- **arguments**: [`$1`: log message to encode and send, `$2`: not used in the function]
- **outputs**: Sends the URL-encoded log message to the gateway server.
- **returns**: No return value, but will have side-effects (sending the log message).
- **example usage**: `remote_log "This is a test log message"`

2.83.3 Quality and security recommendations

Below are some suggested improvements for the `remote_log` function to ensure better quality and security:

1. Validation for the message and the global variables should be added to ensure they are not empty and have valid values.
2. Error handling could be improved – currently, errors (e.g., from curl) are silently ignored.
3. The function currently operates under the assumption that `curl` is available on the system, a check for dependency on curl should be added.
4. The hardcoded URL within the function could potentially pose as a security risk. Consider securing/tidying up this by reading from a secure config file or an environment variable.
5. Data integrity check: Verify the sent message was received/intact at the server side for data reliability.
6. Add debug logging for troubleshooting potential issues.
7. Consider switching to using HTTPS for sending messages to the server to encrypt the data during transmission.

2.84 `rocky_latest_version`

Contained in `lib/functions.d/iso-functions.sh`

2.84.1 Function overview

This bash function, `rocky_latest_version`, fetches the page content from the Rocky Linux download page and parses it to find the latest version number of Rocky Linux available for download. The function leverages `curl`, `grep`, `sed`, and `sort` tools to

fetch and parse HTML content, and extract version numbers from it. If successful, it prints the most recent version number to the standard output.

2.84.2 Technical description

Function: `rocky_latest_version` - **Name**: rocky_latest_version - **Description**: This function fetches and prints the latest version number of Rocky Linux available for download from the official website. - **Globals**: None - **Arguments**: None - **Outputs**: The most recent version number of Rocky Linux, or nothing in case of an error. - **Returns**: 0 if a version number was found and echoed, 1 if the download failed or no version number could be fetched. - **Example usage**:

```
latest_version=$(rocky_latest_version)
echo "The latest version of Rocky Linux is $latest_version"
```

2.84.3 Quality and security recommendations

1. **Error handling and reporting**: Currently, when an error happens (e.g., download fails), the function just returns 1 without any explanation. A more user-friendly approach would be to also print a meaningful error message to the standard error.
2. **Be more specific with `grep` usage**: The function uses a rather broad regular expression to match version numbers. If the page structure changes in the future, it might return wrong results. Instead, consider using a more specific pattern or a different method to get the version number.
3. **Check for `curl` installation**: The function doesn't check if `curl` is installed on the system.
4. **Secure protocol**: The URL is hardcoded with a secure protocol "https". This is good as it ensures secure transmission.
5. **Use of piping and subprocesses**: The function uses multiple pipes and subprocesses. While this is generally acceptable in a bash script, it might negatively impact the performance and also lead to unexpected results if not done carefully.

2.85 `script_render_template`

Contained in `lib/functions.d/kickstart-functions.sh`

2.85.1 Function Overview

The `script_render_template` is a bash function that iterates through all defined environment variables and then evaluates its values by replacing the placeholders present within the template. The placeholders are in the format `@...@` and their respective values would be `${...}`. This handy script allows dynamic content injection into predefined templates through environment variables.

2.85.2 Technical Description

Following is the technical description of the function `script_render_template()`:

- **Name:** script_render_template()
- **Description:** Parses a template, replacing placeholders with corresponding environment variable values.
- **Globals:** No global variables being used.
- **Arguments:** No function level arguments are being passed.
- **Outputs:** The template text with all placeholders replaced with corresponding environment variable values.
- **Returns:** This function does not have return statements.
- **Example Usage:**

```
VAR1="Hello"
VAR2="World"
echo "@VAR1@, @VAR2@!" | script_render_template
# Output: Hello, World!
```

2.85.3 Quality and Security Recommendations

- Validate the variable values before replacing in the template to ensure they do not contain malicious code.
- Handle the cases when an environment variable is not defined gracefully. Currently, it simply replaces with empty string.
- Introduce error handling or exceptions for potential awk failures.
- Secure the script from potential injection vulnerabilities.
- Try to document each segment of the code for better maintainability and understanding.
- Include a way to escape '@' for cases when we do not want replacement.
- Test this script with different use-cases and validate the integrity and security.

2.86 `select_cluster`

Contained in `lib/functions.d/cluster-functions.sh`

2.86.1 Function overview

The `select_cluster` function is designed to allow users to choose from a list of cluster configurations stored in a base directory. The function will catch and handle the case where no clusters are found in the specified directory.

2.86.2 Technical description

- **name:** `select_cluster`
- **description:** This function presents the user with a choice of directories located within a base directory. These directories are intended to represent various clusters from which the user can select. The function uses a conditional check to handle cases where no directories (representing clusters) are found within the specified base directory.
- **globals:** [`HPS_CLUSTER_CONFIG_BASE_DIR`: This global variable stores the location of the base directory containing the clusters.]
- **arguments:** [None]
- **outputs:** If no clusters are found, the function outputs an error message to stderr. In the event that clusters are found, the user is prompted to select one and then the selected directory (cluster) is output.
- **returns:** The function will return `1` if no clusters are found. If a selection is made, the function returns the selection before exiting.
- **example usage:** Assuming `HPS_CLUSTER_CONFIG_BASE_DIR` is a directory with subdirectories representing clusters, the following command will initialize the function:

```
select_cluster
```

2.86.3 Quality and security recommendations

- Consider adding input validation or error handling for the base directory's value to increase the function's robustness.
- Document the expected structure and format of the base directory and its subdirectories.
- Account for potential edge cases, such as the base directory containing non-directory files or the presence of hidden directories.
- Increase the verbosity of error messages to aid in debugging.
- To enhance security, consider restricting the permissions of the base directory and its contents so they are only accessible by intended users or groups.
- If possible, avoid using global variables and favor parameters or local variables within the function; this can help ensure that the function doesn't inadvertently modify data that it doesn't own.

2.87 `set_active_cluster`

Contained in `lib/functions.d/cluster-functions.sh`

2.87.1 Function Overview

The `set_active_cluster` function is used to set a specified cluster as the active one in the system. It receives the cluster name as the argument and forms the cluster directory accordingly, linking it as the active cluster. This function also checks the existence of the cluster directory and the `cluster.conf` file to ensure correct input and functionality.

2.87.2 Technical Description

- **Name**: set_active_cluster
- **Description**: This function sets the active cluster for a given system. It achieves this by creating a symbolic link to the specified cluster's directory from a base directory. Prior to linking, it verifies the existence of the cluster directory and a `cluster.conf` file within.
- **Globals**:
  - `HPS_CLUSTER_CONFIG_BASE_DIR`: the base directory for cluster configurations
- **Arguments**:
  - `$1`: `cluster_name`, the name of the cluster to be set as active
- **Outputs**:
  - If the cluster directory does not exist: outputs an error message "[x] Cluster directory not found: $cluster_dir"
  - If the `cluster.conf` file does not exist: outputs an error message "[x] cluster.conf not found in: $cluster_dir"
  - On successful execution: outputs success message "[OK] Active cluster set to: $cluster_name"
- **Returns**:
  - returns 1 if the cluster directory does not exist
  - returns 2 if the cluster.conf file does not exist
- **Example Usage**:

```
set_active_cluster "test-cluster"
```

2.87.3 Quality and Security Recommendations

1. Ensure that the `cluster_name` argument is sanitized and only accepts valid names, thereby preventing directory traversal attacks.
2. Check the write permissions on the base directory and gracefully handle scenarios where the function does not have the requisite permissions.
3. The function currently does not handle cases where the symbolic link operation fails. Error handling for such scenarios with meaningful message would improve the robustness of the script.
4. Consider encrypting important/confidential data present in the `cluster.conf` file, as the path to it could potentially be exposed.

2.88 ui_clear_screen

Contained in lib/functions.d/cli-ui.sh

2.88.1 Function Overview

The ui_clear_screen() function can be utilized to clear the terminal screen. The function mainly leverages either clear command or the ANSI escape sequence to accomplish this.

2.88.2 Technical Description

Name:
ui_clear_screen

Description:
The ui_clear_screen is a bash function, which is primarily used to clear the terminal screen. It first tries to execute the clear command and if the clear fails, then it executes the printf command with an argument to reset the terminal display.

Globals:
None

Arguments:
None

Outputs:
The terminal screen is reset, providing a clear screen to the user.

Returns:
Doesn't return a value.

Example Usage:

```
ui_clear_screen
```

2.88.3 Quality and Security Recommendations

- Add error handling: For robustness, consider adding an error message to be printed if both clear and printf commands fail.
- Use full paths for commands: To make the script more secure, use the full paths for system commands like clear and printf. This can prevent command hijacking.
- Although the printf is generally safe, it can potentially be misused. If additional functionality is added in the future, ensure that string arguments passed to printf are not user-supplied to avoid possible command injection vulnerabilities.
- Check whether the terminal supports the operations: For better compatibility, one could check whether the terminal supports clearing the screen and the escape sequence "\033c" before executing them.

2.89 __ui_log

Contained in `lib/functions.d/cli-ui.sh`

2.89.1 1. Function overview

The function `__ui_log` is a simple bash function whose primary purpose is to log messages to the standard error output stream (stderr). The messages are prefixed with `[UI]` to indicate the context/source of the log, which in this case is the user interface (UI).

2.89.2 2. Technical description

Function Name: `__ui_log`

Description: A bash function used for logging messages to the stderr with a UI prefix.

Globals: * None

Arguments: * `$*` - Represents all arguments passed to the function. Each argument is treated as a separate word or string.

Outputs: * Outputs the string `[UI]` followed by the string arguments passed to the function to stderr.

Returns: * Does not return any values.

Example Usage:

```
__ui_log "This is a log message"
# Output: [UI] This is a log message
```

2.89.3 3. Quality and security recommendations

- Use a more descriptive name for the function. The current name `__ui_log` does not clearly describe the purpose of the function.
- Include error handling in the function. Check whether the correct number and type of arguments are passed to the function before proceeding with the logging.
- Since the function writes to stderr, consider if all situations where the function will be used are truly error conditions. If not, consider writing non-error output to stdout instead.
- Use consistent formatting and indenting throughout your code to aid readability.
- Be cautious about the information you choose to log. If this function is used to log sensitive information, it could pose a security risk.
- Validate and sanitize input arguments to prevent command injection attacks.

2.90 ui_menu_select

Contained in `lib/functions.d/cli-ui.sh`

2.90.1 Function overview

The `ui_menu_select()` function is a user interactive function aimed to efficiently and easily handle user interface menus in Bash scripts. It displays a menu on the console provided by an input array and repeatedly asks the user to make an input selection until the user makes a valid selection.

2.90.2 Technical description

Name: `ui_menu_select()`

Description: This function presents a user interface on the console for an array-based menu. It accepts an array of options as arguments, presents them as numbered choices to the user, and prompts the user for their selection until a valid selection is made.

Globals: *None*

Arguments: - `$1`: This is the first argument passed to the function, which is used here as the prompt message for the menu. - `shift`: This Bash built-in command shifts the command-line arguments to one position left, essentially removing the first argument `prompt`. - `"${@}"`: This refers to all the arguments passed after array elements, which here are the other options for the user to select.

Outputs: This function will output the selected choice once a valid selection is made.

Returns: The function will return 0 when a valid selection has been made, otherwise it continues to prompt the user for a valid selection.

Example usage:

```
options=("Option 1" "Option 2" "Option 3")
ui_menu_select "Please choose an option:" "${options[@]}"
```

2.90.3 Quality and security recommendations

1. Add validation to ensure reasonable limits on menu options (avoid large number of options).
2. Display an error and exit if no options are provided.
3. Improve error handling. For example, handle errors in non-integer inputs and strings.
4. Enhance usability by handling additional keyboard inputs (such as arrow keys for selection).
5. To minimize injection vulnerabilities, avoid using `eval` or similar commands. Ensure correct quoting and word separation in variable and function usage.

6. Consider a timeout for inputs to prevent indefinite hanging of the script due to inactivity.

2.91 ui_pause

Contained in lib/functions.d/cli-ui.sh

2.91.1 Function overview

The ui_pause() function is written in Bash and it is used as a mechanism to pause a script's execution until the user presses the [Enter] key. This function is vital when testing scripts or ensuring that the user reviews information before the script proceeds.

2.91.2 Technical description

- **Name:** ui_pause()
- **Description:** This Bash function uses the read built-in command combined with -rp option. It will print the prompt "Press [Enter] to continue…" on the standard output (typically on the terminal screen) and then read a line from the standard input (typically from the keyboard). It waits until the user presses the [Enter] key.
- **Globals:** None
- **Arguments:** None
- **Outputs:** It outputs a "Press [Enter] to continue…" prompt message to the terminal screen.
- **Returns:** It does not return a value.
- **Example usage:**

```
#!/bin/bash
echo "This is an important message"
ui_pause
echo "Script resumes here..."
```

2.91.3 Quality and security recommendations

1. This function always assumes the script wants to pause, it may be made more flexible by having a flag or option to toggle this behaviour.
2. Ensure that the prompt message displayed is clear and useful to the user.
3. For better readability, put the common pause message to a separate constant.
4. Check if the script is running in an interactive shell before pausing. This function can cause automated scripts to hang indefinitely if they encounter a pause.
5. It's important to use -r option with read command to prevent it from interpreting any backslashes. Always apply best practice to maintain security.

2.92 ui_print_header

Contained in `lib/functions.d/cli-ui.sh`

2.92.1 1. Function overview

The function `ui_print_header` is a bash shell function written to print a header in the console. This function takes one argument (a string) that is the title of the header. The function prints an empty line, followed by a line filled with equal symbols ("==="), then the title indent by three spaces, and finally another line of equal symbols.

2.92.2 2. Technical description

- **name**: `ui_print_header`
- **description**: This is a bash function designed to print a string as a console header, surrounded above and below by a line of equals signs ("==="). A blank line is printed before the header for readability.
- **globals**: None.
- **arguments**:
  - `$1`: title (string). This is the title to be printed as a console header.
- **outputs**: A formatted console header, surrounded by lines of equals signs and offset by a newline at the beginning.
- **returns**: Nothing is returned by this function.
- **example usage**:

  ```
  ui_print_header "My Custom Header"
  ```

This will output:

```
==================================
   My Custom Header
==================================
```

2.92.3 3. Quality and security recommendations

- Ensure that the input is properly sanitized and is a string to prevent code injection attacks.
- Consider adding error handling to check the validity of the input. If a non-string or a null/empty string is passed as a parameter, the function should have defined behavior.
- Optional: Make the number of "=" dynamic or user-controllable for flexibility.
- Use clear, descriptive names for your functions and parameters. This helps other developers easily understand and use your code.

- Follow a consistent indentation and formatting structure. This improves readability and allows for easier maintenance.
- Always comment your code. While the function may seem straightforward now, proper commenting can help others understand your thought process and makes it easier for future you and others to maintain and update.

2.93 ui_prompt_text

Contained in `lib/functions.d/cli-ui.sh`

2.93.1 Function overview

The `ui_prompt_text` function is a Bash script function that presents a command line prompt to the user and receives textual input as a response. It supports a default text response that will be the output if the user doesn't type anything before hitting the Enter key.

2.93.2 Technical description

Name:
`ui_prompt_text`

Description:
This function echoes a prompt to the user, waits for their response and returns the input they enter. If the user gives no input, the function will use a default value.

Globals:
No globals are used.

Arguments:
- `$1: prompt` - The prompt the user will be presented with. - `$2: default` - The default value that will be used if the user gives no input.

Outputs:
The user's input or the default value if no input was given.

Returns:
The function does not return any value as it directly echoes to standard output.

Example Usage:

```
$ ui_prompt_text "Please enter your name" "Anonymous"
Please enter your name [Anonymous]: <User Input>
```

2.93.3 Quality and security recommendations

- In terms of quality, it may be beneficial to add an option for including a flag that will specify whether to capture the input silently (such as for passwords).

- Adding validation code to enforce input content type would improve function robustness.
- From a security perspective, the script should ideally not echo very sensitive default values back to the user if this script's output can be seen/accessed by others.
- Input could be sanitized to prevent code injection vulnerabilities.

2.94 `ui_prompt_yesno`

Contained in `lib/functions.d/cli-ui.sh`

2.94.1 Function overview

The function `ui_prompt_yesno` is a Bash function designed to prompt a user for a 'yes' or 'no' input. It displays a prompt message, awaits user input, and repeats the prompt until valid input (either 'y' or 'n') is provided. This function supports having a default response which is used when the user simply hits Enter without providing any explicit input.

2.94.2 Technical description

Name: ui_prompt_yesno

Description:
This is a Bash function that incessantly prompts users with a message until they provide a 'Yes' or 'No' input, in the form of 'y' or 'n'. The function supports a default response. The default response becomes the input if a user doesn't provide any and simply hits the Enter key.

Globals:
None

Arguments: - $1: `prompt` - The message to display to the user. - $2: `default` - The default response which becomes the answer if no explicit entry is provided by the user.

Outputs:
Prints the prompt message on the console, potentially multiple times if the user keeps providing invalid input.

Returns: - 0 if the response is 'y' or 'Y'. - 1 if the response is 'n' or 'N'.

Example Usage:

```
if ui_prompt_yesno "Do you wish to continue"; then
  echo "Yes chosen."
else
```

```
  echo "No chosen."
fi
```

2.94.3 Quality and security recommendations

1. Input validation: This function already validates the user input, only accepting 'y' or 'n'. This trial process continues until a valid response is received.
2. Error handling: While the function is resilient against invalid inputs, it could potentially be stuck in an infinite loop if, for example, user input is piped from a non-interactive source. To fix this, a maximum retry count might be added.
3. Local variables: The function uses `local` to narrow down the scope of the variables `prompt`, `default`, and `response`, which is a good practice. However, ensure that your script is consistent in the use of the `local` keyword for variables.
4. Default Response: The function uses a default response 'y'. This can be parameterized based on function call instead of hardcoding it in functionality.

2.95 unmount_distro_iso

Contained in `lib/functions.d/iso-functions.sh`

2.95.1 Function overview

The `unmount_distro_iso` function is used for unmounting distribution ISO files in Linux operating system. It takes in a string that defines the distribution and unmounts the corresponding ISO file. A log message is generated if the distribution ISO is not mounted or fails to unmount.

2.95.2 Technical description

Name: `unmount_distro_iso`

Description: This function unmounts an ISO file of a given Linux distribution. It checks if the distribution ISO is already mounted, if not it logs an info. If it is, it unmounts the ISO file and logs the unmount operation's status.

Globals: - `HPS_DISTROS_DIR`: Directory path of distributions

Arguments: - `$1`: Name of the Linux distribution to unmount - `$2`: Not used in this function

Outputs: Logs info messages relating to the mount point and unmount operation status.

Returns: Returns 0 if the mount point was not mounted or if the dismount was successful. Returns 1 if it fails to unmount the distribution ISO.

Example usage: `unmount_distro_iso ubuntu-20`

2.95.3 Quality and security recommendations

- Always verify if the DISTRO_STRING argument has been provided before using it
- It would be a good idea to handle other error cases, like permission denied or directory not found
- Consider checking for potential security vulnerabilities, like command injection, since file and directory paths are used as an argument to system commands
- Utilize clear and concise logging messages to aid in future debugging or incident response.
- Enforce the principle of least privilege: users should only have permissions to resources they need, this can help limit the potential fallout of a compromised user account.

2.96 update_distro_iso

Contained in lib/functions.d/iso-functions.sh

2.96.1 Function overview

The update_distro_iso function in Bash aims to assist in the handling of a Linux distribution ISO file. It gets the name of a distribution (DISTRO_STRING) as a parameter, specifies the path to the ISO file and its mount point, and checks for the non-existence of the provided distribution name. If the distribution string is available, it will attempt to unmount it. If unmounting fails or the ISO file does not exist, it will display an error and abort the operation. But if it exists, it prompts the user to update the ISO file, and upon the user's confirmation, it attempts to re-mount the ISO file.

2.96.2 Technical description

- **name**: update_distro_iso
- **description**: A Bash function that handles the unmounting, updating, and re-mounting of a Linux distribution ISO file.
- **globals**: [HPS_DISTROS_DIR: A global variable that holds the root directory for multiple Linux distributions.]
- **arguments**: [$1: Distinctive string representing a Linux distribution, typically in the format <CPU>-<MFR>-<OSNAME>-<OSVER>]
- **outputs**: Outputs to stdout, mainly informing the user about the status of the ISO file, like whether it is mounted/unmounted, or whether the user can update the ISO file or not.
- **returns**: Returns 1 if the DISTRO_STRING is not provided, the iso-file of the given distribution is not found or mounting or unmounting fails.

- **example usage**:

  ```
  update_distro_iso ARM-Manufacturer-OSname-OSversion
  ```

2.96.3 Quality and security recommendations

- Always guard against file path injection. Validate inputs thoroughly, and consider if they might include relative path specifiers that could lead to improper filesystem access.
- The function uses the local keyword which makes the variable only visible within the function. Also, keep in mind that even when declaring local variables, if not initialized, they can inherit the value of a global variable of the same name, so be careful with variable initialization.
- Always handle errors correctly. It is beneficial to exit when something goes wrong, rather than continuing on and possibly causing more problems down the line. In this script, if any problem occurs, it returns a non-zero value.
- Consider making the output more user friendly, especially when it comes to error messages. Where possible, provide hints or suggestions for the user. For example, if the ISO file is not found, consider suggesting where the user could find a valid ISO.

2.97 `url_decode`

Contained in `lib/functions.d/hps_log.sh`

2.97.1 Function overview

The `url_decode` function in this Bash script is used to decode a URL. It replaces every '+' in the URL with a space, and all URL-encoded values with their actual ASCII character representation. In case of a failure to write to a specified log file, it logs an error message.

2.97.2 Technical description

Function name: `url_decode`

Description: The function decodes a URL where all '+' are replaced with spaces and all URL-encoded values are replaced with their actual ASCII character representations. After the message is decoded, it's sent to syslog and an attempt is made to write it to a specific log file.

Globals: None.

Arguments: - `$1` (string): The URL that needs to be decoded.

Outputs: Formatted messages that are sent to syslog, and attempts to write a specified log file.

Returns: No return value.

Example usage:

```
msg="$(url_decode "Hello+world%21")"
echo $msg  # prints "Hello world!"
```

2.97.3 Quality and security recommendations

- **Input Validation:** Always check and validate the input that's passed to the function. In this case, make sure that the URL passed to the function meets your requirements or standards.
- **Exception Handling:** Add an else statement to the 'if' condition that checks if the logfile is writable. In the else block, provide instructions on what to do if the logfile isn't writable.
- **Logging:** Always log the start and end of the function in addition to logging failure events.
- **Security:** Be aware of security issues when working with URL decoding, as users with harmful intentions can use percent encoding to evade security filters with ease.
- **Secure Logging:** Ensure logs don't contain sensitive information. If that isn't possible, then make sure that your logs are stored in a secure location and are properly protected.

2.98 `urlencode`

Contained in `lib/functions.d/cgi-functions.sh`

2.98.1 Function overview

The `urlencode` function is used for encoding a URL by converting all non-alphanumeric characters to percent-encoded format. It substitutes a two-digit hexadecimal code, preceded by a percent sign (%), for each disallowed character in the input string.

2.98.2 Technical description

- **Name:** `urlencode`
- **Description:** This function accepts a string as input and generates a URL-encoded version of the string. This is achieved by iterating over each character in the string & testing whether it's alphanumeric or not. Alphanumeric characters are

left untouched, while all other characters are converted into hexadecimal format preceded by a '%' sign.

- **Globals:** None.
- **Arguments:**
  - `$1`: This is the string that has to be URL encoded.
- **Outputs:** The function outputs the URL-encoded string.
- **Returns:** The `urlencode` function doesn't have a return value. It directly outputs the encoded string .
- **Example usage:**

```
url="Hello World"
urlencode "$url"
```

2.98.3 Quality and security recommendations

1. Consider handling edge cases where the input string is empty or contains only spaces.
2. Perform input validation to ensure the input is a string.
3. Although this function may not be vulnerable to significant security risks, always use best practices when dealing with URLs to prevent other types of potential security issues like SQL injections and Cross Site Scripting (XSS).
4. Incorporate error handling to deal with unexpected failures during execution. This can make the function more robust and easier to debug.
5. Add comments to the code to ease understanding and maintenance.
6. Write unit tests for the function to ensure it works correctly with different types of inputs.

2.99 `verify_checksum_signature`

Contained in `lib/functions.d/iso-functions.sh`

2.99.1 Function overview

The `verify_checksum_signature` function provides verification capabilities for downloaded ISO files. It checks the existence of an ISO file specific to a given CPU, manufacturer, OS name, and OS version from a local directory. If the ISO file exists, it cross verifies the ISO's checksum using a remote CHECKSUM file and its GPG signature. The function currently only supports Rocky Linux and reports if the required verification methods for other operating systems are not implemented. It also handles cleanup of temporary files used during the process.

2.99.2 Technical description

- **Name**: `verify_checksum_signature`
- **Description**: The function checks that an ISO file exists and verifies the signature and the checksum of the ISO file. If the checksum or signature do not match the expected values, the function returns an error. The function currently is specifically designed to handle the checksum verification process for Rocky Linux.
- **Globals**: `[ HPS_DISTROS_DIR: The directory to find the iso files ]`
- **Arguments**: `[ $1: The architecture of the CPU, $2: The manufacturer of the CPU, $3: The name of the os, $4: The version of the os ]`
- **Outputs**: The function provides console outputs for each stage of the process, and detailed reports when errors are encountered.
- **Returns**: It can return 0 if the operation was successful i.e. the ISO exists and its checksum and signature match. It returns 1 if the ISO file wasn't found or if the hashes or signatures don't match, or if the verification process hasn't been implemented for the specified operating system.
- **Example usage**:

```
verify_checksum_signature "x86_64" "Intel" "rockylinux" "8.5"
```

2.99.3 Quality and security recommendations

1. Implement error handling for failed curl operations to robustly handle external dependencies.
2. Extend the validation for other operating systems beyond Rocky Linux.
3. Consider allowing the specification of external file paths as input, which would increase the function's flexibility in accessing different directories.
4. Use central configuration for external URLs to ease maintenance.
5. Consider adding a manual override option to bypass the verification process when needed.

2.100 `verify_required_repo_packages`

Contained in `lib/functions.d/repo-functions.sh`

2.100.1 1. Function Overview

This function `verify_required_repo_packages()` is used in a Linux system to validate the existence of specified packages within a given software repository. The function initiates by taking in the repository path as well as the names of necessary

packages as arguments. It validates the existence of the repository path and prompts an error log if the path does not exist. The function then iterates through the list of required packages, checking their existence in the repository. If any package is missing, its name is captured in a 'missing' array. An error log is prompted if any required packages are not found in the repository. If all packages exist in the repository, a success log message is printed.

2.100.2 2. Technical Description

- **Name**: `verify_required_repo_packages`
- **Description**: This function verifies the presence of required packages in a specified software repository.
- **Globals**: None.
- **Arguments**:
  - `$1: repo_path` - The path to the software repository.
  - `$2: required_packages` - An array encapsulating the names of the necessary packages.
- **Outputs**: Logs indicating either of the following; The absence of the repo_path or any required_packages, or a success message indicating all required packages are in the specified repository.
- **Returns**:
  - `1`: When the repository path is not provided or doesn't exist.
  - `2`: When any required package(s) is not found in the repository.
  - `0`: When all required packages are found in the repository.
- **Example Usage**:

```
verify_required_repo_packages "/path/to/repo" "package1"
 ↪  "package2"
```

2.100.3 3. Quality and Security Recommendations

- It is suggested to add more robust error handling. Currently, the function only checks for the existence of the directory and packages, and it might be beneficial to ensure correct permissions or ownership.
- Incorporate a package version specification functionality.
- It is recommended to sanitize all inputs. The function currently trusts its input, which could make it vulnerable to directory traversal attacks if it gets called with untrusted data.
- Use the `-r` (read) flag with the `local` command during variable assignment to prevent field splitting and globbing.
- Quote all variable expansions to avoid word splitting and globbing.
- Provide the explicit path to outside commands such as `find` and `grep` to avoid potential PATH hijacking.

2.101 verify_rocky_checksum_signature

Contained in lib/functions.d/iso-functions.sh

2.101.1 Function Overview

The function verify_rocky_checksum_signature verifies the authenticity of the downloaded Rocky Linux ISO files by checking the expected checksum with the actual checksum. It starts by specifying the version number and architecture (currently limited to x86_64). Several file paths are then defined, including the base URL for the files, the targets directory, and paths for the checksum and signature files.

The checksum and its signature are downloaded, and the Rocky Linux public GPG (GNU Privacy Guard) key is imported. The GPG signature for the checksum file is then verified. If verification passes, the same checksum is used to confirm the accuracy and integrity of the downloaded ISO files. If verification fails at any point, the function returns an error code and a message detailing at what point the verification process stopped.

2.101.2 Technical Description

- **name:** verify_rocky_checksum_signature
- **description:** Verifies the checksum and its signature for the specified Rocky Linux ISO. If verification is successful, checks the expected checksum with the actual checksum for the ISO.
- **globals:** HPS_DISTROS_DIR : This variable sets the base directory where various Linux distribution ISOs will be stored.
- **arguments:** $1: version : The version number of the Rocky Linux distribution whose checksum should be verified.
- **outputs:** Echoes the steps of the process and the results of each verification.
- **returns:** 0 if the every step (including the signature and checksum checks) pass. 1 if the GPG key import fails. 2 if signature verification fails. 3 if no matching checksum can be found for the ISO name. 4 if the actual ISO checksum does not match the expected checksum.
- **example usage:** verify_rocky_checksum_signature 8

2.101.3 Quality and Security Recommendations

1. To improve security, consider using more secure encryption methods or addition of password or passphrase for GPG keys.
2. Implement a strategy for handling updating GPG keys.
3. Expand the function to handle different processor architectures other than “x86_64”.
4. In terms of quality of code, consider refactor or externalize the repeated curl commands into a separate function.

5. If the external resources (e.g., public keys) move or are renamed, the script will fail. Employ an error-handling mechanism for unavailable resources.
6. Validate user input to ensure it fits expected parameters. For instance, verifying version number is a positive integer.

2.102 write_cluster_config

Contained in `lib/functions.d/cluster-functions.sh`

2.102.1 Function overview

The function `write_cluster_config` writes the array of strings passed as arguments into a specified target file. The target file location is provided as the first argument and the subsequent arguments are the values to be written in the target file. If no array of strings is provided, it returns an error message and ends execution.

2.102.2 Technical description

- **name**: `write_cluster_config`
- **description**: This function writes a cluster configuration to a target file. It firstly checks if the array of configuration values is not empty and, if so, writes the configuration values to the target file.
- **globals**: None
- **arguments**:
  - $1: The location of the target file where the configuration values will be written
  - $2: An array of configuration values to be written to the file
- **outputs**: Echo messages indicating the status of operation and configuration values being written.
- **returns**: Returns 1 if the array of configuration values is empty otherwise returns nothing
- **example usage**:

```
values=("value1" "value2" "value3")
write_cluster_config "/path/to/target_file" "${values[@]}"
```

2.102.3 Quality and security recommendations

1. Include input validation: The function should include input validation to ensure the target file path and configuration values comply with expected formats.
2. Use secure methods for file handling: To prevent any security vulnerability, use secure methods for file manipulation.
3. Error handling: Include robust error handling after operations that may fail, like file write operation.

4. Logging: Consider adding logging for tracking and debugging purposes
5. Provide clear and descriptive messages for end users.