OPEN DIGITAL
CONSULTING CO

Platform documentation

HOX project

Stuart J Mackintosh

Monday 11 August 2025

Contents

1 HPS system documentation

1.1 Overview

- **Project introduction** – Purpose of HPS, architecture, intended use cases.
- **Design decisions** – Records of key technical choices and rationale.
- **System components** – Description of major directories and modules.

1.2 Quick start

- **Prerequisites** – Required host operating system, packages, and network setup.
- **Installation** – Deploying `hps-container` with `hps-system` and `hps-config`.
- **First cluster setup** – Initialising a cluster with `cluster-configure.sh`.
- **Booting a host** – PXE boot process and selecting a host profile.
- **Verification** – Checking service status and logs.

1.3 System administration

- **Directory layout** – Locations for configuration, logs, distributions, and packages.
- **Cluster management** – Creating, switching, and editing clusters.
- **Host management** – Adding, removing, and updating host configurations.
- **Distribution and repository management** – Managing ISOs, PXE trees, and package repositories.
- **Service management** – Starting, stopping, and reloading dnsmasq, nginx, and supervisord.
- **Backup and restore** – Protecting and recovering configuration and repository data.

1.4 Functions reference

- Automatically generated from `lib/functions.d/` and other libraries.
- One function per page with purpose, arguments, usage examples, and related functions.

1.5 Advanced configuration

- **Kickstart and preseed templates** – Structure, variables, and customisation.
- **iPXE menus** – How menus are built and extended.
- **Storage provisioning** – SCH node disk detection, reporting, and configuration.
- **Integration points** – Hooks for OpenSVC, monitoring, and external systems.

1.6 Troubleshooting

- **PXE boot issues** – Common causes and fixes.
- **Service failures** – Diagnosing and restarting services.
- **Distribution and repository problems** – Checksums, GPG keys, and synchronisation errors.
- **Network problems** – DHCP conflicts, VLAN configuration, and firewall blocks.

1.7 Development

- **Code layout** – File structure and conventions for scripts and libraries.
- **Adding functions** – Naming, argument handling, logging, and documentation guidelines.
- **Testing** – Using `cli/test.sh` and other test harnesses.
- **Contributing** – Workflow, coding standards, and submission process.

1.8 Appendices

- **Glossary** – Definitions of terms and acronyms used in HPS.
- **Environment variables** – Description of exported variables and their use.
- **Decision records** – Full list of design decisions.
- **Reference configurations** – Example cluster, host, and service configuration files.

2 Overview

This chapter introduces the HPS system, its purpose, core architecture, and intended use cases.
It also records the major design decisions that shape the system and defines terminology used throughout.

2.1 Design decisions

Stub: Summarise key design decisions. Link to full decision records in the reference section.

2.2 Introduction

Stub: Provide a high-level explanation of HPS, its goals, and where it fits into the broader infrastructure platform.

2.3 Terminology

Stub: Define common terms, acronyms, and abbreviations used across the documentation.

3 Quick start

A fast path to installing HPS, configuring a cluster, and booting a node.
This section covers the essentials and assumes no prior HPS experience.

3.1 Booting a host

Stub: PXE boot process overview and selecting a host profile.

3.2 First cluster setup

Stub: Use `cluster-configure.sh` to create the first cluster and set its parameters.

3.3 Installation

Stub: Step-by-step guide to deploy `hps-container` with `hps-system` and `hps-config`.

3.4 Prerequisites

Stub: List hardware, OS, packages, network setup, and permissions needed before starting.

3.5 Verification

Stub: Confirming services are active and hosts are provisioned correctly.

4 Installing HPS

How to install the HPS system on the provisioning node, configure services, and verify readiness.

4.1 Hardware

- choosing

4.2 Dependencies and prerequisites

- Operating system ISO’s
- Designing networking and numbering

4.3 Obtaining HPS

Stub: How to acquire HPS source, container, and configuration files.

4.4 Service verification

Stub: Checking that dnsmasq, nginx, supervisord, and other components are running.

4.5 Upgrades and maintenance

Stub: Keeping `hps-system` updated without overwriting configuration files.

5 Deploying and configuring a cluster with nodes

Creating a cluster, configuring its settings, provisioning nodes, and verifying the environment.

5.1 Cluster configuration

Stub: Setting DHCP interface, storage subnets, OS type, and other cluster settings.

5.2 Cluster creation

Stub: Running `cluster-configure.sh` and choosing cluster parameters.

5.3 Distribution management

Stub: Adding ISOs, extracting PXE trees, and maintaining package repositories.

5.4 Host profiles

Stub: Assigning profiles such as SCH, TCH, DRH, and CCH to nodes.

5.5 Node provisioning

Stub: PXE/iPXE boot workflow and automated node configuration.

5.6 Service management

Stub: Controlling dnsmasq, nginx, supervisord, and other HPS services.

5.7 Verification

Stub: Checking that nodes are deployed correctly and services are operational.

6 Deploying the disaster recovery node

Installing and configuring the DR node, synchronising data, and testing recovery procedures.

6.1 Deployment process

Stub: Steps for installing and integrating the DR node into the cluster.

6.2 Purpose of the DR node

Stub: Explain the role of the DR node in ensuring service continuity.

6.3 Failover and recovery testing

Stub: Procedures for verifying DR readiness and simulating failover scenarios.

6.4 Preparation

Stub: Hardware, storage, and network prerequisites for the DR node.

6.5 Synchronisation

Stub: Methods for keeping DR node data in sync with the primary environment.

6.5.1 HPS system documentation

6.5.1.1 Overview

- **Project introduction** – Purpose of HPS, architecture, intended use cases.
- **Design decisions** – Records of key technical choices and rationale.
- **System components** – Description of major directories and modules.

6.5.1.2 Quick start

- **Prerequisites** – Required host operating system, packages, and network setup.
- **Installation** – Deploying `hps-container` with `hps-system` and `hps-config`.
- **First cluster setup** – Initialising a cluster with `cluster-configure.sh`.
- **Booting a host** – PXE boot process and selecting a host profile.
- **Verification** – Checking service status and logs.

6.5.1.3 System administration

- **Directory layout** – Locations for configuration, logs, distributions, and packages.
- **Cluster management** – Creating, switching, and editing clusters.
- **Host management** – Adding, removing, and updating host configurations.
- **Distribution and repository management** – Managing ISOs, PXE trees, and package repositories.
- **Service management** – Starting, stopping, and reloading dnsmasq, nginx, and supervisord.
- **Backup and restore** – Protecting and recovering configuration and repository data.

6.5.1.4 Functions reference

- Automatically generated from `lib/functions.d/` and other libraries.
- One function per page with purpose, arguments, usage examples, and related functions.

6.5.1.5 Advanced configuration

- **Kickstart and preseed templates** – Structure, variables, and customisation.
- **iPXE menus** – How menus are built and extended.
- **Storage provisioning** – SCH node disk detection, reporting, and configuration.
- **Integration points** – Hooks for OpenSVC, monitoring, and external systems.

6.5.1.6 Troubleshooting

- **PXE boot issues** – Common causes and fixes.
- **Service failures** – Diagnosing and restarting services.
- **Distribution and repository problems** – Checksums, GPG keys, and synchronisation errors.
- **Network problems** – DHCP conflicts, VLAN configuration, and firewall blocks.

6.5.1.7 Development

- **Code layout** – File structure and conventions for scripts and libraries.
- **Adding functions** – Naming, argument handling, logging, and documentation guidelines.
- **Testing** – Using `cli/test.sh` and other test harnesses.
- **Contributing** – Workflow, coding standards, and submission process.

6.5.1.8 Appendices

- **Glossary** – Definitions of terms and acronyms used in HPS.
- **Environment variables** – Description of exported variables and their use.
- **Decision records** – Full list of design decisions.
- **Reference configurations** – Example cluster, host, and service configuration files.

6.5.1.9 `__ui_log`

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

Function signature: d3d49681e2b691a5ff908ab2cfc80feb43c6f2c7f2aa6c60521377957fc9244b

Function overview The `__ui_log()` function in Bash is a utility function designed to generate log outputs. The function captures all function parameters as input, formats them in a specific log output pattern “[UI] {parameters}”, and redirects this formatted log to the standard error (stderr) stream. This function is particularly helpful in debugging or tracing execution in scripts due to its concise structure and easy application.

Technical description

- **Name:** __ui_log()
- **Description:** A Bash function that takes several parameters, formats them in a standard log output pattern “[UI] {parameters}”, and redirects the output to stderr.
  - **Globals:** None.
  - **Arguments:** [$*]: All arguments are accepted and processed as one string. This characteristic allows the function to take and output an arbitrary list of parameters.
  - **Outputs:** Formatted log “[UI] {parameters}” to standard error.
  - **Returns:** No values are returned.
  - **Example usage:** `bash __ui_log "This is a test log entry"` This call will print “[UI] This is a test log entry” to the standard error output.

Quality and security recommendations

1. Considering the input is directly displayed in a log, ensure that any sensitive data or user data is kept private or obfuscated before invoking the `__ui_log()` function.
2. Add input validation and error handling to ensure calls to `__ui_log()` are working as expected.
3. Standardize the log output pattern to improve logging efficiency and analysis.
4. Consider redirecting output to a dedicated log file instead of stderr to avoid cluttering the error stream and make logs more manageable.

6.6 Functions reference

Bash functions with descriptions, arguments, and examples.

6.6.0.1 `bootstrap_initialise_distro`

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

Function signature: 1664d8a7eb2277600c48a3bc6974c34ea586df3fb5c9bfb5b7b8268285d91c63

Function Overview `bootstrap_initialise_distro` is a Bash function used to bootstrap initialization for a particular Linux distribution. This process involves setting up the baseline environment for the chosen OS. The function processes the commands from the provisioning server offline, avoiding networking issues and increasing the stability of the overall process.

Technical Description

- **Name:** `bootstrap_initialise_distro`
- **Description:** This function takes a MAC address as an input and initializes the desired Linux distribution operating system for the device associated with that MAC address. The initialization process is done offline by transmitting commands from a provisioning server.
- **Globals:** None
- **Arguments:** [$1: The MAC address of the target device]
- **Outputs:** The function outputs a Bash script that is sent to standard output. This output can be saved and executed on the target device for initializing the distribution.
- **Returns:** This function doesn't have a return value.
- **Example usage:** `bash bootstrap_initialise_distro "00:1B:44:11:3A:B7"`

Quality and Security Recommendations

1. **Input Validation:** Validate the MAC address input to ensure it's in the correct format and not a potential injection attack.

2. **Use of the Local Keyword:** Ensure that variables used within the function are declared with the local keyword to prevent shadowing of variables from the outer scope.
3. **Error Handling:** Implement error handling mechanism to capture any failures during the offline loading of the Bash script from the provisioning server.
4. **Secure Transmission:** Although the initialization process is done offline, ensure the Bash script transmission from the provisioning server is secure to prevent any potential script tampering.
5. **Logging:** Include logging within the function to ensure any errors or issues can be traced post-execution.

6.6.0.2 `build_yum_repo`

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

Function signature: 55fba2752d840b45670dcc10f8be084d3234f9c87a37a1959fcde6a9247be828

Function overview `build_yum_repo` is a bash function that is used to build a YUM repository in a specified path. It first performs a series of checks to ensure that the provided repository path is valid and that necessary resources are available, specifically the `createrepo_c` command. The function then checks if there have been changes in the RPM files within the repository folder. If there are changes or no previous state, it then executes `createrepo_c` to generate a new repository. Finally, it saves the current state and logs the success of the operation.

Technical description

- **Name:** build_yum_repo
- **Description:** A bash function to build a YUM repository in a specified directory.
- **Globals:** [HPS_PACKAGES_DIR: The directory where packages for the server are stored, DIST_STRING: String representing the distribution of packages or repository]
- **Arguments:** [$1: repo_path, path to create the YUM repository in]
- **Outputs:** Log messages about the process and possible errors.
- **Returns:** 0 if the function successfully ran ‘createrepo_c’, 1 if the repo path does not exist or wasn’t provided, 2 if ‘createrepo_c’ command wasn’t found.
- **Example usage:** `build_yum_repo "$HPS_PACKAGES_DIR/$DIST_STRING/Repo"`

Quality and security recommendations

1. One key area of improvement would be to add input validation for the ‘repo_path’ parameter to ensure it conforms to the expected pattern of a valid path.
2. It would be beneficial to standardize the logging function to output the logs in a consistent format, making it easier to troubleshoot.

3. Implementing error handling logic for the command `createrepo_c --update "$repo_path"` would be beneficial to ensure that the function can recover from unexpected situations or provide useful feedback in case of failure.
4. Security-wise, consider reviewing the permissions and setting restrictive permissions to the .rpm files in the repo_path.
5. Consider using full paths to commands to avoid potential issues with PATH hijacking.

6.6.0.3 `cgi_header_plain`

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

Function signature: cce7eb1544681966ec11d5f298135158497fc2ac56c89b00c63c84b8e1bc733a

Function Overview The provided Bash function, `cgi_header_plain`, is used to send a plain text HTTP header to the client. This informs the user agent that the content that follows will be in plain text format. It does so by first delivering the `Content-Type: text/plain` MIME type header, then sending an additional echo statement to insert a newline. This effectively separates the headers from the subsequent content.

Technical Description The `cgi_header_plain` function can be broken down into the following components:

- **Name:** `cgi_header_plain`
- **Description:** This function generates a plain text HTTP header to notify the client that the upcoming content will be in the plaintext format.
- **Globals:** None
- **Arguments:** None
- **Outputs:** `Content-Type: text/plain`
- **Returns:** None
- **Example Usage:**

```bash
#!/bin/bash
cgi_header_plain
echo "This is the body of the response."
```

In this example, the `cgi_header_plain` function will output a plain content-type HTTP header, followed by a newline. The echo statement then provides the body of the response text.

Quality and Security Recommendations

1. **Input Validation:** Since this function does not accept any arguments and does not handle user-supplied input, there are no input validation concerns in this context.

2. **Error Handling:** Although this function does not perform any actions that might result in an error (like file I/O or network calls), in a more complex function, appropriate error handling should be put in place to handle potential failures.
3. **Documentation:** The function has no inline comments. Although the function is straightforward, consistent inline comments can help to make the codebase more maintainable and easier to understand by other developers.
4. **Security:** This function emits a static response, and therefore does not interface with sensitive data, limiting the potential for security vulnerabilities.

6.6.0.4 `cgi_log`

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

Function signature: 9f2c2cf7c0d57e85a08611717b5d691eddf235f096bbc311bf9d58541f0c77b3

Function Overview The `cgi_log` function is a bash function designed to create logs for a cgi program in the ipxe system. This function acquires a message as an input and appends this message to a `cgi.log` file located in the `/var/log/ipxe/` directory. The function also adds a timestamp in front of every message log.

Technical Description **Name:** `cgi_log`

Description: The `cgi_log` function takes a message (string input), adds a timestamp to it, and appends it to the `cgi.log` file found in the `/var/log/ipxe/` directory.

Globals: None

Arguments: `$1: msg` (The message string that is to be logged)

Outputs: Appends the timestamped message to a file (`cgi.log`).

Returns: Nothing.

Example Usage:

```
cgi_log "This is a log message."
```

Last command will append the following log to the `/var/log/ipxe/cgi.log` file:

```
[date in "%F %T" format] This is a log message.
```

Quality and Security Recommendations

1. Before appending a message to the `cgi.log` file, ensure that the file exists and has the correct permissions to avoid potential file not found exceptions.
2. Validate the input message to prevent logging of potentially harmful scripts or commands.
3. In order to prevent potential log file overflow, implement log rotation strategies.

4. Always make sure to redirect both `stdout` and `stderr` to capture any kind of execution message for full logging.
5. For better security considerations, avoid disclosing sensitive information in the log. Details such as user credentials or personal details should be anonymized or completely left out.
6. It's recommended to wrap this function in a `try-catch` or similar error handling method to handle any possible errors effectively and maintain the stability of your running script.

6.6.0.5 `cgi_param`

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

Function signature: 5007f95c313c04a01df7ba39bff0241f44511cd570d795bc11a890edf032f323

Function overview The `cgi_param` function parses the `QUERY_STRING` once and then performs some action based on the provided command (`$cmd`). The possible commands are `get`, `exists`, and `equals`. The `get` command prints the value of the parameter with the provided key (`$key`). The `exists` command checks if the given key exists. The `equals` command verifies if the value of the given key is equal to the provided value (`$value`). If the command is not recognized, the function prints an error message and returns 2.

Technical description **Function**: `cgi_param`

Description: The function parses the `QUERY_STRING` to obtain CGI parameters only once, and perform actions (`get`, `exists`, `equals`) on these parameters depending on the command provided.

Globals: [`__CGI_PARAMS_PARSED`: A flag used to ensure that the `QUERY_STRING` is parsed only once, `CGI_PARAMS`: An associative array that holds the parsed CGI parameters]

Arguments: - `$1(cmd)`: The command to be executed on the CGI parameters. It can be `get`, `exists`, or `equals`. - `$2(key)`: The key of the CGI parameter to be processed. - `$3(value)`: The value that is supposed to be compared with the value of the CGI parameter specified by `$key` in case of `equals` command.

Outputs: Depending on the `cmd` argument, the function might: - print the value of the `$key` parameter (`get` command), - print an error message when an unrecognized command is provided.

Returns: - Nothing, in scenarios where the function checks whether a certain parameter exists. - 2 when an invalid command is provided.

Example Usage:

```
cgi_param get username
cgi_param exists userpassword
cgi_param equals sessionId 12345
```

Quality and security recommendations

1. Consider using more strict error handling: Currently, the function only handles unrecognized commands but does not cover cases where other arguments might be missing or provided in an incorrect format.
2. Sanitize all input: Before passing the `QUERY_STRING` to the `read` command, ensure it doesn't contain any malicious data that can lead to command injection or other vulnerabilities.
3. Validate parsed parameters: Besides checking the parameters' format before storing them in the `CGI_PARAMS` array, it would be beneficial to also check their content (e.g., make sure the values are within expected boundaries/standards for specific keys).
4. Document expected format and restrictions for `QUERY_STRING`, `$cmd`, `$key`, and `$value`: Having a clear explanation on how the function expects its input, and how it handles unexpected input, can help prevent misuse and make debugging easier.

6.6.0.6 `cgi_success`

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

Function signature: 3c468a0d5bf1a1d432f4efcb2a108889f0d0e762f542f50c29b92350f02de3bc

Function overview The `cgi_success` function is a simple Bash function implemented within the context of a CGI (Common Gateway Interface) script to display a success message. This function first calls the `cgi_header_plain` function to set the headers for the CGI script and then prints out whatever string is passed as the first argument to the function.

Technical description **Name**: `cgi_success`

Description: This function is used to display a success message in CGI scripts. It first sets the necessary headers by calling the `cgi_header_plain` function and then prints out the string value that is passed as the first argument to the function.

Globals: None.

Arguments: - `$1: string`. The string that will be printed out when the function is called.

Outputs: This function outputs the string value passed as an argument to standard output.

Returns: Nothing explicit.

Example usage: `cgi_success "Operation completed successfully"` This will output the string "Operation completed successfully" to the standard output.

Quality and security recommendations

1. **Input Validation**: Ensure that the input passed to the `cgi_success` function is correctly validated and sanitized to prevent potential cross-site scripting (XSS) vulnerabilities.
2. **Error Handling**: Include error handling to capture and manage any situations where the `cgi_header_plain` function fails.
3. **Documentation**: Maintain comprehensive function-level comments in the code to facilitate the understanding of the function's operation and usage.
4. **Testing**: Include this function within the unit testing framework to ensure it behaves as expected over different kinds of input data.
5. **Use Escaping Functions**: If output includes some special characters (like "<" or "&"), use the respective escape function before outputting them to prevent any conflicts or issues.
6. **Check Return Status**: After calling `cgi_header_plain`, check its return status to ensure it worked as expected before proceeding.

6.6.0.7 `check_and_download_latest_rocky`

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

Function signature: ac86f8c78c7149b4fbc3125a2f52dd65b160ad01f437b5eeb5b504bd1a90c6b1

Function Overview The `check_and_download_latest_rocky` function in Bash is used to check for and download the latest version of Rocky Linux in ISO format. It will first check for the latest version available at a defined URL and then proceed to download it if the ISO does not already exist in the specified directory. Additionally, it sends the ISO for extraction to support PXE booting.

Technical Description

- Function name: `check_and_download_latest_rocky`
- Description: This function, written in Bash, checks for the latest version of Rocky Linux at the provided URL, downloads it as an ISO file if it does not exist in the user defined directory, and sends the ISO for extraction to support PXE boot functionalities.
- Globals: [`HPS_DISTROS_DIR`: this is the directory where the downloaded ISOs are stored]
- Arguments: None

- Outputs: Echo statements informing the user of the current process (check for latest version, downloading the ISO or finding it already present in the directory).
- Returns: The function does not explicitly return a value. However, the effect of function is the Rocky Linux ISO being downloaded and stored in the user defined directory.
- Example Usage: `check_and_download_latest_rocky`

Quality and Security Recommendations

1. The function does not currently have any error handing built in. It is recommended to add error handling steps to make this function more robust and user-friendly.
2. As a good practice for bash scripts, it is suggested to quote your variables as to avoid unexpected behavior from word splitting and pathname expansion. For example, use `"$iso_path"` instead of `$iso_path`.
3. Make download URL, architecture, and other variables taking static values parameters to the function to make it more reusable and flexible.
4. Replace echo statements for user feedback with a proper and more detailed logging system for easier debugging and traceability.
5. Always validate the downloaded ISO to ensure its integrity and safety. This function currently may download an ISO but does not complete any kind of validity or safety check on it.
6. Secure your curl download with the appropriate security flags. The flags `--fail --show-error --location` are not enough to ensure a secure download.

6.6.0.8 `check_latest_version`

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

Function signature: 3b874dd0168548a5363b9c63f357dab1016d452e1155213b1190930b50bb44c5

Function overview This Bash function, `check_latest_version`, checks the latest version of an Operating System (OS) based on parameters passed to the function. Specifically, it needs the CPU architecture, the manufacturer, and the OS name. If the OS is Rocky Linux, it fetches the HTML of the Rocky Linux downloads page and greps for the version number in the HTML. It retrieves all versions, sorts them in descending order, and outputs the latest version. If the OS variant is unknown, the function returns an error.

Technical description

- **Name**: `check_latest_version`
- **Description**: The function checks and outputs the latest version of an Operating System (OS) from its downloads page based on the input parameters. It works specifically with Rocky Linux.
- **Globals**: None.

- **Arguments**:
  - `$1`: CPU architecture (Description of the CPU architecture)
  - `$2`: Manufacturer (Description of the manufacturer)
  - `$3`: OS Name (Name of the Operating System)
- **Outputs**:
  - Error message if it fails to fetch the download page or the OS variant is unknown
  - Message stating the latest version of the operating system
- **Returns**:
  - `1` If it fails to fetch the download page, finds no versions for the OS, or if the OS variant is unknown
  - `0` If it successfully gets the latest version of the OS
- **Example Usage**:

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

Quality and security recommendations

1. Curl is operating without a timeout, which can cause the function to hang indefinitely if the server fails to respond. Incorporating a reasonable timeout can prevent this.
2. The function should validate the identifiers (`$cpu`, `$mfr`, and `$osname`) against known values before proceeding for extra security.
3. Error handling/messages can be improved to provide more specific and meaningful information for debugging purposes.
4. SSL verification is turned off in the curl command. This can be a security risk.
5. The function should handle other OS types, not just Rocky Linux, to make it more versatile.
6. The regular expression in the `grep` command could be improved to better match version patterns and avoid false positives.
7. The function currently runs on a set URL, which makes it less flexible. Adding ability to handle different URLs will make the function more reusable.

6.6.0.9 `cluster_config`

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

Function signature: ea561bb0141a57e976a8d270b800ed25635238facc99a0dcb1b5ee58af4ad106

Function Overview The function `cluster_config()` provides utility for getting, setting, and checking the existence of keys in the active cluster configuration file. It takes in three arguments: the operation type (`get`, `set`, or `exists`), the key whose value to interact with, and optionally, the value to update the key to (for `set` operation).

Technical Description

- **Name:** `cluster_config()`
- **Description:** The `cluster_config()` is a function for managing keys in the active cluster configuration file. It contains operations for getting the values of keys (`get`), for setting the values of keys (`set`), and for checking if keys exist within the file (`exists`).
- **Globals:** None
- **Arguments:** [`$1`: operation (get, set, exists), `$2`: Key for the operation, `$3`: Optional value for the `set` operation]
- **Outputs:** Echoes either the value of the key (for `get` operation), the result of a key existence check (for `exists` operation), or error messages for inappropriate key operation or missing active cluster configuration file.
- **Returns:** Returns `1` if no active cluster configuration is found or `2` if an unknown operation is passed.
- **Example Usage:**

  ```
  cluster_config get foo
  cluster_config set foo bar
  cluster_config exists foo
  ```

Quality and Security Recommendations

1. Implement input validation to prevent attempts of injection or any other form of command compromise through the function parameters.
2. Improve error handling by providing more specific error messages and varying the return codes for different error types to assist in troubleshooting.
3. Consider making it more dynamic by allowing interaction with more than one cluster configuration file.
4. Provide support for other text file formats, or secure the cluster configuration file to prevent unauthorized or unintended modifications.

6.6.0.10 `cluster_has_installed_sch`

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

Function signature: befb7e577a31bf8e64c1179ffa1c6cd4d2ec9a30913e1c6b26986c37fd0762cc

1. Function Overview This function, `cluster_has_installed_sch`, checks if there is a Software Configuration (SCH) installed in a cluster. It does this by reading through the configuration files located in a specified directory. If found one with a `TYPE` of `SCH` and a `STATE` of `INSTALLED`, the function will stop inspecting other files and return with a success status (0). If it doesn't find any, it will return with a failure status (1).

2. Technical Description

- **Name:** `cluster_has_installed_sch`
- **Description:** This function validates if a software configuration (SCH) is installed in a cluster by checking the configuration files in a certain directory.
- **Globals:** [`HPS_HOST_CONFIG_DIR`: This variable is containing the path to the configuration directory]
- **Arguments:** No arguments needed for this function
- **Outputs:** Prints nothing to stdout.
- **Returns:** 0 if SCH is installed, 1 otherwise.
- **Example Usage:**

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

3. Quality and Security Recommendations

1. Use clear and understandable variable names and add comments to improve code readability.
2. Add error handling to handle cases when the configuration directory does not exist or is not readable.
3. Avoid using globals when possible and pass them as arguments to the function instead. This promotes better encapsulation and reuse of the function.
4. For a more defensive programming approach, validate that config_file actually contains a valid configuration file.
5. Be careful to protect against potential code injections via the `val` or `key` variables when they are being used in other scripts or functions.

6.6.0.11 `configure_dnsmasq`

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

Function signature: 44a71126e0bd635025c26058732db02e7d8f2e8e1619d95b58e5ea4cb78813b5

Function Overview The function `configure_dnsmasq` is used to configure the dnsmasq service on a server. Dnsmasq is a lightweight, easy to configure DNS forwarding server and DHCP server. It is designed to provide DNS and DHCP services to a small network. It serves and caches the DNS using a /etc/hosts type file and provides DHCP services.

Technical Description

- **Name**: `configure_dnsmasq`
- **Description**: The function generates a dnsmasq configuration file based on the active cluster configuration and copies iPXE boot files to a pre-specified TFTP directory.
- **Globals**: [`HPS_SERVICE_CONFIG_DIR: Directory where the service configuration files are stored`, `DHCP_IP: DHCP Server IP`, `HPS_TFTP_DIR: Directory for TFTP Server`, `NETWORK_CIDR: Network CIDR for DHCP server`]
- **Arguments**: None
- **Outputs**: If DHCP_IP value is not set, it displays an error message: "[ERROR] No DHCP IP, can't configure dnsmasq". After successful configuration, the message "[OK] dnsmasq config generated at: ${DNSMASQ_CONF}" appears.
- **Returns**: Function exits with status 0 when DHCP_IP isn't set, otherwise it doesn't have a definitive return value.
- **Example Usage**: `configure_dnsmasq`.

Quality and Security Recommendations

1. It is recommended to validate the inputs and file handling parts of the function to prevent any potential security risks such as injection attacks or file overwrites.
2. The function should handle the absence of required binaries such as `cat`, `source` and `mkdir`. An error should be thrown if these are not present.
3. Potential failures in the commands within the function should be handled gracefully with meaningful error messages to the user.
4. Return a status code from the function that can be evaluated by the caller to decide the success or failure of the function. This will make it more useful in scripting.
5. It is recommended to use a secure method to store and retrieve the network details instead of sourcing them from a file for more security.
6. Run this script with necessary privileges only, as it can alter system configurations.

6.6.0.12 `configure_ipxe`

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

Function signature: 8be018f219f7bb96b1582cebaa58986ae6fca49645ba4a4141473836d7e5add6

Function overview The `configure_ipxe` function is for setting up the IPXE boot configuration. It constructs an IPXE script file called 'boot.ipxe' that carries out the following operations after booting:

- Obtains an IP address via DHCP
- Normalises the MAC address
- Attempts to load configuration specific to the host machine from the server.

- If it is unable to find the configuration, it opens a menu to the user for manual configuration.
- The function can handle different host configuration types which include Thin Compute Host, Storage Cluster Host, Disaster Recovery Host, and Container Cluster Host.

Technical description

- **Name**: `configure_ipxe`
- **Description**: This function is used to create the IPXE boot configuration script file called 'boot.ipxe'.
- **Globals**:
  - `HPS_MENU_CONFIG_DIR`: The directory in which the ipxe boot file will be created.
  - `DHCP_IP`: The IP for the DHCP server.
- **Arguments**: None
- **Outputs**: An IPXE script file in the directory specified by `HPS_MENU_CONFIG_DIR`.
- **Returns**: Outputs a success message once the 'boot.ipxe' file has been successfully created and written to.
- **Example usage**: `configure_ipxe`

Quality and Security Recommendations

1. Always validate any form of user inputs or server responses to protect against SQL injection attacks and XSS attacks.
2. To prevent pathname expansion or word splitting, always quote your variables. Inconsistent quoting can lead to vulnerabilities.
3. Error messages should be written to STDERR instead of STDOUT.
4. It is advisable to perform sanity checks before making a directory to ensure that it does not already exist and that you have sufficient permissions.
5. Set stricter permissions to the created ipxe boot configuration file to prevent unauthorised access or modifications.
6. It may be prudent to clean up or limit the logged information to prevent the disclosure of sensitive information.

6.6.0.13 `configure_kickstart`

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

Function signature: b9c974579a4cf0918b1f523ab5546c0fabf4f4fe0d6a0a4ab77a23765ef1cbca

Function Overview The `configure_kickstart` function is used to generate a Kickstart file that can be utilized for automated installations of the CentOS operating system on Linux clusters. The function creates a Kickstart file with a predefined configuration, where cluster name is passed as an argument. If the required argument is not provided, the function outputs an error message and exits. Once the Kickstart file is generated, the function confirms its creation by outputting its location.

Technical Description

- **Name**: configure_kickstart
- **Description**: This function generates a Kickstart file for automated CentOS installations on Linux clusters.
- **Globals**: [CLUSTER_NAME: The name of the cluster, KICKSTART_PATH: The path where the generated Kickstart file is stored]
- **Arguments**: [$1: The name of the cluster to be used in the Kickstart file creation]
- **Outputs**: Confirmation of Kickstart file generation and its location, or an error message if arguments are not provided adequately.
- **Returns**: Returns 1 and exits in case no argument is provided, otherwise no explicit return value.
- **Example Usage**: `configure_kickstart cluster1`

Quality and Security Recommendations

1. It is a good practice to conduct variable and input validation for better quality code execution.
2. Avoid using plaintext passwords in scripts or configuration files as seen in this Kickstart file. It's better to use hashed passwords or some form of secure password management.
3. The initial root password is locked, it would be good to establish a process for initial login or provide mechanisms for first-time initialization of root password.
4. Logs for the post-install scripts should be redirected to a secure location where access is strictly controlled to prevent unauthorized access.
5. Further secure the script by providing restrictive file permissions. The generated Kickstart file may contain sensitive information, hence permissions should be carefully managed.

6.6.0.14 `configure_nginx`

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

Function signature: c4c2d185045e9971e0c4fc289b25486b51e2b27e21851e55ee76e0dcdfafce20

1. Function Overview The function `configure_nginx()` is a bash function that configures NGINX in a system. It first sources the filename of the active cluster, suppresses any error that might crop, and then sets up the path for NGINX configuration. Lastly, it writes a new configuration into the NGINX configuration file.

2. Technical Description

- `name`: `configure_nginx()`
- `description`: This function sets up the NGINX configurations by sourcing the active cluster file and writing into NGINX configuration file.
- `globals`: [`HPS_SERVICE_CONFIG_DIR`: the directory where the service configuration is stored]
- `arguments`: [`$1`: Not applicable in this function, `$2`: Not applicable]
- `outputs`: A written NGINX configuration file in the directory specified by `HPS_SERVICE_CONFIG_DIR`.
- `returns`: Nothing, as the function performs an operation but does not return any value.
- `example usage`: simply call the function in the script as `configure_nginx`

3. Quality and Security Recommendations

1. Always ensure that the path used in the `source` command is correct to avoid sourcing the wrong file which may lead to problems.
2. Ensure proper permissions for creating and writing to the NGINX configuration file. This is to prevent unauthorized access and modifications.
3. Keep track of the global variables and avoid using them unnecessarily to decrease coupling and increase the function's reusability.
4. Always handle possible errors that may occur during the execution of the function.
5. Always test the function with various inputs to ensure it behaves as expected. Always aim for high test coverage.
6. The function seems to overwrite existing NGINX configuration each time it runs. You should consider preserving and versioning old configurations instead of simply overwriting.

6.6.0.15 `configure_supervisor_core`

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

Function signature: a08c044216c8e9f8b7860f39202cf1d683e9e33a541b75fa8f48512b90a43743

Function overview The `configure_supervisor_core` function is primarily used to generate the Supervisor configuration file (`supervisord.conf`). It firstly ensures the existence of the `/var/log/supervisor` directory, which will hold the Supervisor logs. It then sets up a variety of server settings, notably including the server URL, username, password, default logging level, authentication method, etc. After the configuration file is created, it also prints a confirmation message to the user with the location of the generated file.

Technical description

- **Name:** `configure_supervisor_core`
- **Description:** This function is designed to generate a Supervisor configuration file (`supervisord.conf`). It defines basic server settings, including network configuration, security settings, logging settings, and more. It'll then save these settings into the configuration file.
- **Globals:** `HPS_SERVICE_CONFIG_DIR`: The directory that holds the service configuration file of Home Preserving Sweets.
- **Arguments:** This function does not take any arguments.
- **Outputs:** Outputs a confirmation message, confirming the successful generation of the Supervisor configuration file along with its directory location.
- **Returns:** No return value, as the function operates by side effect.
- **Example Usage:** The function can simply be called using `configure_supervisor_core`.

Quality and security recommendations

1. It might be more efficient and safer to check the existence of `HPS_SERVICE_CONFIG_DIR` before the function begins file writing operations.
2. To improve security, it's advised to not hard-code sensitive credentials like username and password in a system configuration file. Instead, consider retrieving them from a more secure place like an environment variable.
3. To standardize the function, it would be better to return specific values to signify the success or failure of the function, useful for debugging and error handling.
4. The function could benefit from better error handling, such as catching if the directory or file fails to create. This may provide a better user experience, and aids with debugging.
5. The `user=root` in the configuration file may make the system vulnerable. It would be better to run the supervisor process with minimal necessary permissions.

6.6.0.16 `configure_supervisor_services`

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

Function signature: 35009e0251c9cae404f78b96bd139c323154a7be18a41d67cb2334fd6df96c5f

Function overview The `configure_supervisor_services` function is used for configuring Supervisor services. It creates a Supervisor services configuration file at a specified directory and then writes service information for dnsmasq, nginx, and fcgiwrap into the file. Each service is also automatically started and is set to restart itself.

Technical description

- **Name:** `configure_supervisor_services`
- **Description:** Configures Supervisor services by creating and modifying a configuration file for Supervisor with service-related data. Specifically, the function sets up configurations for dnsmasq, nginx, and fcgiwrap services.
- **Globals:** [`SUPERVISORD_CONF`: Path to the Supervisor services configuration file, `HPS_SERVICE_CONFIG_DIR`: Directory containing the service configurations]
- **Arguments:** [None]
- **Outputs:** A Supervisor services configuration file at the value of `SUPERVISORD_CONF`
- **Returns:** No direct return value but outputs a confirmation string after successfully creating the config file
- **Example usage:**

```
configure_supervisor_services
```

Quality and security recommendations

1. Make sure variable `HPS_SERVICE_CONFIG_DIR` is defined before running the function as it is not checked or declared inside the function.
2. Consider validating if the directories and files specified by the `HPS_SERVICE_CONFIG_DIR` and `SUPERVISORD_CONF` variables exist or are writable before attempting to create or modify them.
3. Adding error handling or logging in case of unsuccessful operations such as file creation or writing could enhance the reliability of the function.
4. Check that the configuration files for the individual services are all in the correct format and contain the expected entries to avoid potential errors when they are loaded.

6.6.0.17 `count_clusters`

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

Function signature: f85fd6ef48db5cb9daae832a23896a3098e1015c070794f3fe360262d2c24879

Function overview The function `count_clusters` is designed to count the number of clusters within a specified base directory. Initially, it offers an error message if the specified directory cannot be found. On recognizing the directory, the function utilizes a shell option to prevent errors if no files were found. It aggregates the relevant cluster directories into an array. If no clusters are found, the function provides an appropriate error message and returns 0. If clusters are identified, their count is printed out.

Technical description

- **Name**: count_clusters
- **Description**: This function counts the number of cluster directories within a specified base directory.
- **Globals**: [HPS_CLUSTER_CONFIG_BASE_DIR: It is the path to the configuration base directory for the clusters]
- **Arguments**: [There are no arguments used in this function]
- **Outputs**: Prints out the total number of clusters found within the base directory, or error messages if a base directory or clusters are not found.
- **Returns**: 0 if base directory not found or no clusters found, otherwise it does not return any explicit value but prints the count of clusters.
- **Example usage**: `bash export HPS_CLUSTER_CONFIG_BASE_DIR=/path/to/clusters count_clusters`

Quality and security recommendations

1. Add proper input validation for the `HPS_CLUSTER_CONFIG_BASE_DIR` variable. This would help to mitigate potential issues in the event of unauthorized access or erroneous input.
2. Handle all potential error messages or exceptions for better function robustness.
3. Always ensure that user-provided paths do not allow for path traversal – sanitize the inputs.
4. Use more robust file handling and manipulation commands to explore directories, providing for a more universally applicable function.

6.6.0.18 `detect_storage_devices`

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

Function signature: 26407abf67962b945819ec70c2d91c7a26b60c144eeb209e22298b013ba307ed

Function Overview The function `detect_storage_devices()` is used to retrieve and display data related to all block devices available on a system, such as name, model, vendor, serial number, bus type, device type, size, usage, and speed. It makes use of other utility functions to get this data and puts it into a designated output string, each set of data separated by `---`.

Technical Description

- **Name**: detect_storage_devices

- **Description**: This function retrieves details of all block storage devices on a system including device name, model, vendor, serial number, bus type, device type, size, usage and speed.
- **Globals**: None
- **Arguments**: None
- **Outputs**: Displays details of every block device on output.
- **Returns**: Doesn't return any value.
- **Example usage**: `detect_storage_devices`

Quality and Security Recommendations

1. Implement input validation to ensure that only valid block storage devices are being processed.
2. Handle exceptions when calling other utility functions (like `get_device_model`, etc.) to ensure the function doesn't break due to unexpected errors.
3. Secure the information being displayed by the function, as it might contain sensitive data like device serial number.
4. Consider optimizing the function if the number of storage devices is high to avoid performance issues.

6.6.0.19 `download_iso`

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

Function signature: a769efc53df917e64e3dbdfb8acb70dff4b4cb4a89efd3b55a689c47cde86e91

Function overview The function `download_iso` downloads an ISO image of a specific Operating System version for a given CPU architecture from the Internet. It stores the downloaded ISO image in a predefined directory and name. Currently, it only supports the `rockylinux` Operating System variant. If an ISO image that matches the provided parameters already exists in the target directory, the function does not attempt to download it again, instead serving the existing copy.

Technical description

- **name**: download_iso
- **description**: A Bash function that downloads an ISO image from the web and saves it in a specific local directory.
- **globals**: `N/A`
- **arguments**: [
  - `$1`: CPU - The architecture of the CPU (e.g., x86_64),
  - `$2`: MFR - Manufacturer identifier (currently not used in the function),
  - `$3`: OSNAME - The name of the Operating System variant,
  - `$4`: OSVER - The version of the Operating System]

- **outputs**: Prints messages about the status of the executed operations
- **returns**:
  - `0` if function executes successfully.
  - `1` if unsupported OS variant is provided or if the function fails to download the ISO image.
- **example usage**: `download_iso x86_64 intel rockylinux 10`

Quality and Security Recommendations

1. The function should validate its input parameters to ensure they adhere to a predetermined format or regex, preventing potential errors in filename construction.
2. The function should support more OS variants, or perhaps have a mechanism for easily adding and managing support for new OS variants.
3. Currently, the function does not perform any action with the `mfr` argument. It should either be used or removed to avoid confusion.
4. The function could include functionality to handle various error states more gracefully. For example, if the directory creation or ISO download process fails, it should capture the exact error and possibly attempt to resolve it.
5. Add checksum verification for ISO images after download for enhanced integrity and security assurance.
6. Since the function seems to be designed to handle sensitive data (Operating System ISO images), it could incorporate some mechanisms for security-hardening, such as encryption of the downloaded ISO image.

6.6.0.20 `export_dynamic_paths`

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

Function signature: a6b2a47e08ed460524c491151fd944d515572f0fe9d26a1a2d5d310ad72b99b5

Function Overview The function `export_dynamic_paths` is a Bash function used to export different environment paths for a specific cluster. It takes at least one argument - `cluster_name`. If no cluster name is given, it would utilize the name of the currently active cluster.

The base directory defaults to `/srv/hps-config/clusters` unless the environment variable `HPS_CLUSTER_CONFIG_BASE_DIR` is present with a different directory. This function then exports differing paths including `CLUSTER_NAME`, `HPS_CLUSTER_CONFIG_DIR`, and `HPS_HOST_CONFIG_DIR`.

Technical Description

- **name**: `export_dynamic_paths`
- **description**: A Bash function which exports environment paths for a specific cluster.

- **globals**: [`HPS_CLUSTER_CONFIG_BASE_DIR`: A global variable specifying base directory. Defaults to `/srv/hps-config/clusters` if not provided]
- **arguments**: [`$1`: `cluster_name`, The name of the cluster for which specific paths are exported]
- **outputs**: Sets the environment variables `CLUSTER_NAME`, `HPS_CLUSTER_CONFIG_DIR`, and `HPS_HOST_CONFIG_DIR`.
- **returns**: The function returns `1` when there is no active cluster and none is specified. In normal operation, it returns `0`.
- **example usage**:

```
export_dynamic_paths "cluster_1"
echo $CLUSTER_NAME
echo $HPS_CLUSTER_CONFIG_DIR
echo $HPS_HOST_CONFIG_DIR
```

This would export the cluster paths for the cluster named "cluster_1".

Quality and Security Recommendations

1. This function should validate the input for `cluster_name`, possibly for null, special characters or invalid cluster names.
2. Use stricter condition checking. For example, in the `if` condition that checks whether `cluster_name` is empty, consider checking for a null string or whitespace.
3. Ensure that the directory being referred to in variables exist and has the correct permissions, ensure correct error handling for this.
4. Ensure correct permissions are set on the exported paths to avoid unnecessary access from unauthorized users.
5. Use echo statements or a logging function for more verbose output to aid in debugging.
6. Add more error checking, like checking if changing of the environmental variables succeeded.

6.6.0.21 `extract_iso_for_pxe`

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

Function signature: 291539ccd925935805dd8c7d2a52eeb587e5f13f95dab12edd40bdbe8cf2a210

Function Overview The `extract_iso_for_pxe` function is designed to extract ISO files for network booting via PXE. The function accepts four parameters: a CPU type, a manufacturer ID, an operating system name, and its version. It assumes that the ISO files are stored in a specific directory structure, and uses this information to construct the path to the ISO file and the path to the desired extraction directory. If the ISO file is not found, an error is reported. If it has already been extracted, a success message is displayed. If not, the ISO file is extracted to the directory. If the extraction fails, an error is reported, otherwise a success message is displayed.

Technical Description

- **Name**: extract_iso_for_pxe
- **Description**: This function extracts ISO files for PXE (Preboot Execution Environment), using specified parameters to locate the ISO file and choose the extraction directory.
- **Globals**:
  - `HPS_DISTROS_DIR`: Base directory for ISO file extraction.
- **Arguments**:
  - `$1: cpu`: Type of CPU.
  - `$2: mfr`: Manufacturer ID.
  - `$3: osname`: Name of the operating system.
  - `$4: osver`: Version of the operating system.
- **Outputs**:
  - Error or success messages depending on whether the ISO file is found, whether it has already been extracted, and whether the extraction succeeds.
- **Returns**:
  - `1` if the ISO file is not found or the extraction fails.
  - `0` if the ISO file is successfully extracted or if it had already been extracted.
- **Example usage**: `extract_iso_for_pxe "intel" "dell" "ubuntu" "18.04"`

Quality and Security Recommendations

1. Add validation of input arguments to check for potential invalid or malicious input.
2. Implement error handling to make the function more robust against potential failures.
3. Secure the directories that the function has write access to, to prevent unintended modification or deletion of files.
4. Test the function in various environments to ensure compatibility.
5. Document the requirements for the ISO directory structure and extraction directory structure to ensure correct usage.

6.6.0.22 `extract_rocky_iso_for_pxe`

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

Function signature: 52e8a2e1e3a65096a2e0e1ac3696f4c21521a7cc3c3b006a7056ede3446b2ada

Function Overview The `extract_rocky_iso_for_pxe` function is designed to automate the process of extracting the ISO image files of Rocky Linux for PXE (Preboot eXecution Environment). This function takes in the path to the ISO file, the Linux version, and the CPU architecture as arguments. The files are then extracted to a specific directory, which is determined by the provided parameters. The function supports both the `bsdtar` and `fuseiso` commands for this extraction process.

Technical Description

- **Name**: `extract_rocky_iso_for_pxe`
- **Description**: Extracts a Rocky Linux ISO for PXE
- **Globals**: `[ HPS_DISTROS_DIR: A directory for distribution versions, CPU: Processor architecture, MFR: Manufacturer (in this case 'linux'), OSNAME: Operating system name (in this case 'rockylinux'), OSVER: Operating system version, extract_dir: Extraction directory path ]`
- **Arguments**: `[ $1: iso_path (Path to ISO file), $2: version (Linux version), $3: CPU (CPU architecture) ]`
- **Outputs**: Extracted ISO file for PXE
- **Returns**: 1 if neither `bsdtar` nor `fuseiso` is found, otherwise nothing
- **Example usage**: `extract_rocky_iso_for_pxe "/path/to/iso" "8" "x86_64"`

Quality and Security Recommendations

1. Add more error checks and possibly introduce handling for different edge cases, such as invalid inputs or read/write errors.
2. The function requires a significant amount of system memory to mount and process the files. Make sure to free up space or manage memory effectively.
3. Always check the source of the ISO image download to ensure it is a trustworthy site.
4. Given the function's role in setting up a booting environment, it's especially important to ensure the integrity and authenticity of the ISO images.
5. Consider integrating a log mechanism to track the performance of the function in case of troubleshooting.
6. Implement a way to check if the created directories already exist to avoid overwriting any previous versions.
7. Include more informative and user-friendly messaging to communicate the function's process and outcomes to the user.
8. Make sure the `HPS_DISTROS_DIR` variable is defined and valid before executing the function.

6.6.0.23 `fetch_and_register_source_file`

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

Function signature: 97650ca173393457a3e6151b2fbb0a997e425c3010fc3c13018364e050618d70

Function Overview The `fetch_and_register_source_file` function, as the name suggests, does two things. Firstly, it fetches the source file from a given location on the internet. The location is given in the form of a URL. This fetching is performed using another function call - `fetch_source_file`. Secondly, the function registers the fetched file into a local system. The registration is performed using the `register_source_file` function. The filename for the registration process is taken as an optional argument. If it isn't provided, the filename is extracted from the URL.

Technical Description

- **Name**: `fetch_and_register_source_file`
- **Description**: This function fetches a source file from a location denoted by a URL, which is the first argument given to the function. It then registers this fetched file to the local system with another function call.
- **Globals**: Not applicable.
- **Arguments**:
  - `$1`: The URL from which the source file will be fetched.
  - `$2`: Handler for where the fetched source file will be registered.
  - `$3`: Optional argument. The filename for the registration process. If not given, it is extracted from the URL.
- **Outputs**: This function does not output anything.
- **Returns**: This function does not return anything.
- **Example usage**:

```
fetch_and_register_source_file "http://example.com/file.tar.gz" myHandler
fetch_and_register_source_file "http://example.com/file.tar.gz" myHandler "arch
```

Quality and Security Recommendations

1. Input validation is quite crucial. This function should validate the URL before trying to fetch the file from it. It can protect the function from potential security vulnerabilities.
2. The function could incorporate error handling to gracefully handle scenarios when it can't fetch a source file or register it.
3. Regarding security, it's recommended that the function verify the integrity of the downloaded file. This could be done by checking for a checksum of a file, and ensuring it matches the expected value.
4. One could enhance this function's functionality by returning error codes when it can't fetch a file or register it. The error codes can help determine the error's exact cause.

6.6.0.24 `fetch_source_file`

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

Function signature: 9851dec43ce9f3e94856800874741f8ff28deda9346709daaa88282763e49999

Function Overview `fetch_source_file` is a Bash function that's primarily responsible for managing the download of files in a Linux system. It takes in a URL of a source file as well as a filename. If the filename isn't provided, it infers the filename from the URL. The file is then downloaded and stored in a specific directory. If the file already exists in the directory, it won't re-download it. The function supports the handling of download errors.

Technical Description

- **Name**: `fetch_source_file`
- **Description**: This function downloads a file from a given URL and places it in a specific directory. The filename can be manually assigned or automatically inferred from the URL. It also handles cases where the file already exists, by skipping the download, and reports any issues encountered during the download process.
- **Globals**: `HPS_PACKAGES_DIR`: Path to where the packages are stored
- **Arguments**:
  - `$1`: `url`: URL of the source file to download.
  - `$2`: `filename`: Name of the file to be created. If not provided, the name is inferred from the URL.
- **Outputs**: Status messages to indicate the download process, including whether the file already exists or if the download failed.
- **Returns**: Returns `0` if the file already exists or if the download was successful; returns `1` if the download fails.
- **Example Usage**:

```
fetch_source_file "http://example.com/sourcefile.tar.gz"
```

Quality and Security Recommendations

1. Always use secure protocols (HTTPS over HTTP) to ensure the file is downloaded securely.
2. Validate the URL for the source file before passing it to this function. Ensure it points to a trusted and reliable source.
3. Consider adding a checksum function to verify the integrity of the downloaded file. This could help detect any corrupted or tampered files.
4. Implement robust error handling. While the function handles download errors, it can further be improved to handle other potential failures like issues with directory creation or file saving.

5. Add logs and more verbose outputs for easier troubleshooting.
6. Consider adding testing to validate the correct behavior of the function.

6.6.0.25 `generate_dhcp_range_simple`

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

Function signature: 0b96ef1d9a801dba7584c3c03ea9f1327da1cd2685d343519ee3ae72aa66ecd8

Function overview The `generate_dhcp_range_simple` function generates a DHCP range in a subnet. It takes a network CIDR, a gateway IP, and an optional count parameter to determine the range size. If the count is not provided, the function defaults to creating a range of 20 IPs.

Technical description

- **Name**: `generate_dhcp_range_simple`
- **Description**: A function that generates the IP range for DHCP in a given subnet.
- **Globals**: None
- **Arguments**: `$1`: network_cidr (e.g 192.168.50.0/24), `$2`: gateway_ip (e.g 192.168.50.1), `$3`: count (optional, defaults to 20)
- **Outputs**: This function outputs the calculated DHCP range.
- **Returns**: None directly. Outputs are sent to stdout.
- **Example Usage**:

  generate_dhcp_range_simple "192.168.50.0/24" "192.168.50.1" 30 This function would generate a DHCP range in the `192.168.50.0/24` subnet, starting at the gateway IP "192.168.50.1" and creating a range of 30 IPs.

Quality and security recommendations

1. The function uses ipcalc, an external tool, to calculate network and broadcast addresses. If this tool is not available or fails, the function will be unable to proceed. You may want to add error checks after calling `ipcalc`.
2. Ensure that the range count does not result in overlapping IP ranges. You may want to add logic to prevent the creation of overlapping ranges.
3. It is a good practice to validate all inputs before proceeding with the function. You may want to add checks to ensure that the network_cidr and gateway_ip are valid IPs in correct format.

6.6.0.26 `generate_ks`

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

Function signature: b973477b6d7653d80151f03150fadcfc0d8f1f85a77f353025a267d5257c8529

Function Overview The `generate_ks` function is primarily used in the generation of a kickstart file for system installations. This function sets numerous environment variables based on the given MAC ID and host type, retrieves several configuration options such as IP, netmask, hostname, and others, and it makes these variables available for the installation script. After setting up all these configurations, it changes the state of the host configuration to "INSTALLING" and finally outputs the installation script by rendering its template.

Technical Description

- **Name:** generate_ks
- **Description:** This Bash function generates a kickstart file for system installations. It accepts a MAC ID and host type, configures certain settings, and prints an installation script.
- **Globals:** [macid: Given MAC ID, HOST_TYPE: Specified type of the host, HOST_IP: IP address of the host, HOST_NETMASK: Subnet mask for the host IP, HOST_NAME: Name of the host, HOST_GATEWAY: Gateway of the host, HOST_DNS: DNS server for the host, HOST_TEMPLATE_DIR: Directory where host installation templates are kept, INSTALLER_TYPE: Type of installer being used, HOST_INSTALL_SCRIPT: Path to the installation script, HOST_INSTALL_FUNC: Installation function based on installer type and host type]
- **Arguments:** [$1: MAC ID of the host, $2: type of the host]
- **Outputs:** Installation script created with the help of the set configurations.
- **Returns:** No return value, as the function primarily performs operations and outputs a script.
- **Example usage:**

```
generate_ks 00:16:3e:4a:2b:4c server
```

Quality and Security Recommendations

1. Consider modifying this function to return error codes in case specific operations fail, this will allow better error handling.
2. To secure the function, ensure that only authorized personnel can call the function within the script.
3. Validate the inputs for MAC ID and host type to prevent injection of malicious scripts.
4. Try to reduce usage of global variables as much as possible.

5. More comments need to be added for easier understanding of the function's workflow.

6.6.0.27 `get_active_cluster_file`

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

Function signature: 688ac6268430b453c3ca87ba80fb648cfbbf4b3b63dabdcbb2f8a76ed8fb685d

Function Overview The function `get_active_cluster_file` serves to return the contents of the active cluster configuration. It operations on a local link (`HPS_CLUSTER_CONFIG_DIR/active-cluster`) that is expected to be a symbolic link to the location of the active cluster config file (`cluster.conf`). It checks if this link exists and resolves to a valid file, in which case it will output the contents of that file. If there are any errors (the link doesn't exist, it cannot be resolved, or the resolved target is not a file), a corresponding error message is echoed and the function return with a status code of `1`.

Technical Description **Name:** `get_active_cluster_file`

Description: This function returns the contents of the active cluster configuration file.

Globals: [`HPS_CLUSTER_CONFIG_DIR`: the directory where the cluster configuration files are stored]

Arguments: [None]

Outputs: Either the contents of the active cluster configuration file, or an error message if something went wrong

Returns: `0` if successful, `1` otherwise

Example Usage:

```
source cluster-utils.sh
get_active_cluster_file
```

Quality and Security Recommendations

1. Use more specific error codes for different error cases (e.g. `1` for missing link, `2` for unresolved link, etc.) for easier diagnosis and handling of different error conditions.
2. Use `realpath` instead of `readlink -f` to resolve symbolic links for better portability (not all systems have `readlink -f`).
3. Consider using `readlink -m` instead of `readlink -f` to resolve symbolic links without failing when the target doesn't exist, as this function might be used in a context where it's acceptable for the target to later be created.

4. Validate the contents of the configuration file before returning them to ensure they match an expected format and prevent potential code injection.
5. To prevent file path injection attacks, ensure that `HPS_CLUSTER_CONFIG_DIR` cannot be modified by untrusted users. Consider making “active-cluster” a constant rather than hardcoding it in multiple places.

6.6.0.28 `get_active_cluster_filename`

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

Function signature: 7207d6053d7feea7b49e85e1ac6488ca00a74177aae1bc317d07cf6d8e044fd0

Function Overview The function `get_active_cluster_filename()` is a Bash function designed to get the name of the active cluster file in a specific configuration directory. At the core, the function uses a symlink pointing to an ‘active cluster’. If the symlink or the target doesn’t exist, error messages are printed to STDERR and the function returns a status of 1. If everything is successful, the function echoes the absolute path of ‘cluster.conf’ in the active cluster directory.

Technical Description

- **Name**: get_active_cluster_filename
- **Description**: This Bash function returns the path to ‘cluster.conf’ in the active cluster directory. The ‘active-cluster’ is a symlink residing in the HPS_CLUSTER_CONFIG_BASE_DIR that points to the active cluster directory. The function handles cases where there is no symlink at the location or if the symlink couldn’t be resolved.
- **Globals**: [HPS_CLUSTER_CONFIG_BASE_DIR: A string containing the base path to the configuration directory]
- **Arguments**: None
- **Outputs**: If successful, function echoes the absolute path of ‘cluster.conf’ in the active cluster directory. Otherwise, printing error messages to STDERR.
- **Returns**: On failure, returns 1
- **Example usage**: `get_active_cluster_filename`

Quality and Security Recommendations

1. Implement additional error checking and handling. For instance, validate that `HPS_CLUSTER_CONFIG_BASE_DIR` contains a valid directory path.
2. Avoid storing or retrieving sensitive information without adequate protections in place. If the cluster configurations contain sensitive information, ensure they’re secured.
3. Implement symlink protection measures to mitigate symlink attacks like race conditions. Attacks could potentially change symlink destinations.

4. Provide more informative error messages that wouldn't leak potentially sensitive information about the file system to unauthorized users.
5. Handle SIGINT (Ctrl+C) event to clean up intermediate outputs or state and exit gracefully.

6.6.0.29 `get_active_cluster_info`

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

Function signature: 9343c2f4e3a1e406a9b3535d1ac5d056322bb64a1c7c08b644dcc2aa5b70914a

Function overview The Bash function `get_active_cluster_info()` is used to retrieve the configuration file of an active cluster. It checks different scenarios: whether there are no clusters, only one cluster, or multiple clusters. Depending on the outcome, it returns a specific cluster configuration file, prints an error message, or prompts the user to select a cluster from a list.

Technical description

- **Name**: `get_active_cluster_info`
- **Description**: This function hosts a script that can get the active configuration of a cluster from a provided directory. The function checks if there are no, one, or multiple active clusters in the directory and acts accordingly.
- **Globals**:
  - `HPS_CLUSTER_CONFIG_BASE_DIR`: The base directory where cluster configurations are stored. Default is `/srv/hps-config/clusters`.
- **Arguments**: None
- **Outputs**: This function prints the path of the active cluster configuration file, or an error message when no clusters are found or when multiple clusters are found but none is designated as active.
- **Returns**: This function returns `0` if it successfully fetches the configuration file of an active cluster. Otherwise, it returns `1` if no clusters are found or when the active cluster configuration file is not found.
- **Example usage**: To fetch the active cluster configuration use the function in the terminal as follows: `bash get_active_cluster_info` ##### Quality and security recommendations

1. The function assumes the base directory which can present a security issue if not handled right. Consider asking for input from the user or having it as an argument with a default option.
2. Error messages are sent to `stderr`, which is good practice, but status codes returned might not be indicative enough for external applications that could use the function.

3. Add comments to code blocks to explain what each section does, which will make maintaining the code easier.
4. Consider edge cases where `readlink -f "$active_link"` might not return the expected output.
5. Make sure permissions for the active link file and directory are set correctly and only accessible by the expected users or groups of users.

6.6.0.30 `get_all_block_devices`

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

Function signature: add7deb6a087d72238984be839fb5488e13aff3ae7251f93f2c1814d79162625

Function Overview The bash function `get_all_block_devices` provides a way to get all block devices in a Linux system. The function reads the `/sys/block` directory, where all block devices are represented. The function filters these objects to only include those with the type of "disk". Each block device appears as a directory in `/sys/block/{device}`, where `{device}` is the name of the block device. The function prints out the names of all devices of the type "disk".

Technical Description

- **name**: `get_all_block_devices`
- **description**: This function scans the `/sys/block` directory for all block devices, filtering out entries that are not of type "disk". It then prints out their names to STDOUT.
- **globals**: None
- **arguments**: No arguments needed
- **outputs**: This function will reveal the names all block devices of type "disk".
- **returns**:
  - Name of all block devices of type disk.
- **example usage**:

```
get_all_block_devices
```

The above example will output a list of all block devices of type 'disk' in your system.

Quality and Security Recommendations

1. Error handling for the `basename` command should be added to ensure the script doesn't crash if it fails.
2. Additional checks can be made to confirm that "/sys/block" exists and is accessible before running the function.
3. Ensure that the function is running with the necessary permissions to access the "/sys/block" directory. If not, the function could fail or return incorrect results.

4. Implement sufficient input validation or sanitation to thwart potential security risks such as command injection.

6.6.0.31 `get_client_mac`

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

Function signature: 6b28946d90a3fddf9e18f05fe426e933a15b243f01dde38c4257f73ffa1f64f3

Function overview The `get_client_mac` function is used to get the MAC (Media Access Control) address of a client machine through its IP address. This function makes use of ARP (Address Resolution Protocol) packets to find the MAC address assigned to an IP. If the initial method fails, it attempts to extract the MAC address using `arp` tool.

Technical description

- **Name**: `get_client_mac`
- **Description**: The function `get_client_mac` discerns and outputs the MAC address of a client machine given its IP address. It first confirms that the IP address is valid. If not, the function returns 1. If the IP address is valid, an ARP request is triggered via a ping request, forcing the target machine to respond and update the ARP table. The function then uses an awk-driven regular expression search within the IP neighbour list to extract the MAC address. On unsuccessful attempts, it tries to extract the MAC address using the `arp` command.
- **Globals**: VAR - No global variables used in the function.
- **Arguments**: `$1` - This argument represents the IP address for which the MAC address is to be fetched.
- **Outputs**: The MAC address associated with the given IP.
- **Returns**: If the supplied IP address is empty, the function will return 1.
- **Example Usage**: `get_client_mac "192.168.1.1"`

Quality and security recommendations

1. Input validation should be more robust, ensuring that only valid IP addresses are accepted for further processing to avoid possible abuse or unexpected behavior.
2. To enhance performance, add a check for the MAC address in the local ARP cache before sending a ping request to the client machine.
3. Using `ping` and `arp` tools might be blocked by firewall rules or network policies. Always check if these tools can reach the client machine.
4. Include error detecting mechanisms to check if the ARP update or ping was successful.
5. The code must carefully handle the absence of the awk or `arp` commands on some systems. If these commands are not available, the function won't work as expected.

6.6.0.32 `get_device_bus_type`

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

Function signature: 71e77c52eb2ba1d481c9ef51c928878b08e2c1088eeb0bf2fbe433c62633a476

Function overview The BASH function `get_device_bus_type` is designed to return the bus type for a given device. The function identifies if the device is NVMe, SSD, or HDD. It does this by first checking if the device is a NVMe type. If it isn't, it checks if the device is SSD. If it's still not identified, the function defaults to identifying the device as HDD.

Technical description

- **Name:** get_device_bus_type
- **Description:** This function takes in a variable name `dev` and checks the device's bus type (NVMe, SSD, or HDD).
- **Globals:** None
- **Arguments:**
  - `$1:` This is the input provided when calling the function. It represents the device to be checked.
- **Outputs:** The function outputs either "NVMe", "SSD" or "HDD" depending on the type of device being checked.
- **Returns:** It does not have a specified return value since the output is made via an echo command.
- **Example usage:**

  ```
  bus_type=$(get_device_bus_type "/dev/nvme0n1")
  echo $bus_type
  ```

Quality and Security recommendations

1. Input Validation: Ensure that the argument provided to the function is valid. This may involve checking that the string provided is not null or empty and does indeed represent a device path on the system.
2. Error Handling: Implement better error handling capabilities to handle scenarios where the device bus type does not match any of the predefined types.
3. Secure Coding Practices: In order to prevent potential field injection or other types of malicious behavior, sanitize all inputs to the function.
4. Testing: Conduct thorough testing of the function under different scenarios for quality assurance.
5. Documentation: Keep this function's documentation up-to-date and detailed with any changes made over time.

6. Code Usage: Ensure secure usage of this function. Do not use it in contexts where its output can be manipulated for harmful outcomes.

6.6.0.33 `get_device_model`

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

Function signature: ec59456e4d8546a811f0f4533daf51da08474fb4ada4f1efb264e30d2fea7091

Function overview The function `get_device_model()` is designed to retrieve information about a specific device in a Linux system. It operates by fetching relevant details from a designated system file, before formatting the output to strip out unnecessary spaces, in the event that device information is unavailable, the function will output “unknown”.

Technical description Definition block for `get_device_model()`

- **name**: `get_device_model()`
- **description**: `get_device_model()` is a function that retrieves and prints the model name of a given device. It removes any unnecessary spaces in the model name. If the model name cannot be fetched, it prints “unknown”.
- **globals**: None
- **arguments**:
  - `$1`: Represents the device identifier
- **outputs**: It prints the device model name to standard output or prints “unknown” if the model name can’t be fetched.
- **returns**: None. The function does not explicitly return a value; the result is printed directly to the standard output.
- **example usage**:

```
get_device_model /dev/sda
```

This command will print the model name of the device `/dev/sda`.

Quality and security recommendations

1. It’s recommended to validate the input to ensure it’s not empty and is a valid device identifier.
2. Avoid using `cat` when you can read files directly into a variable.
3. Errors in the `cat` command are sent to `/dev/null`, thus the user will not be informed about possible issues related to non-existent files or lack of required permissions.
4. The function does not return any status code or error message when a failure occurs. All errors should be handled properly, preferably by returning a unique status code.

5. The function will fail silently if the directory or the file does not exist. It is recommended to check if the file exists before trying to read it.

6.6.0.34 `get_device_rotational`

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

Function signature: 1002aec4163a82a48171eea735ad8700633333695a4b75dcb9ecc3317d14222f

Function overview `get_device_rotational` is a Bash function that gets the rotational characteristic of a block device on a Linux-based system. A parameter, which is the name of a block device, is passed to the function. If the device is rotational, the function will return '1', and if it's not or if there is an error retrieving the information, it will also return '1'. This function is typically used to check if a hard drive is Solid-State Drive (SSD) or Hard Disk Drive (HDD).

Technical description

- **name**: `get_device_rotational`
- **description**: This function checks if a block device is rotational or not in a Linux system. It returns '1' in case of an error.
- **globals**: None
- **arguments**:
  - `$1`: dev - a string representing the block device to check
- **outputs**: The function outputs either '0' (for non-rotational devices) or '1' (for rotational devices and in case of an error).
- **returns**: The function doesn't have a specific return statement, so the last command's status is returned, i.e., the status of 'echo'.
- **example usage**:

```
if [[ "$(get_device_rotational sda)" -eq "0" ]]; then
    echo "Device is an SSD"
fi
```

Quality and security recommendations

1. To increase the robustness of the function, error handling should be expanded. Instead of defaulting to '1' for errors, a different unique value or message should be returned.
2. It's a good practice to validate the input to the function. Ensure the script properly handles cases where the device doesn't exist or the device name provided is invalid.
3. There is potential for a race condition if the device's type changes after the function checks it and before its return is used. Depending on how critical this is, consider ways to avoid this type of situation.

4. Security-wise, make sure the script is being run with appropriate permissions to read the device information.
5. Comment the function adequately to ensure that its use and return cases are clear to other developers.

6.6.0.35 `get_device_serial`

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

Function signature: 8e1b210627973a0cc629e646a16460819b61ae08954c5dba162358f2b6cb4532

1. Function overview The `get_device_serial` function is a bash function designed to retrieve the serial number of a device. This function utilizes the `udevadm` command-line utility and the `grep` and `cut` command to extract the serial number from the device properties.

2. Technical description

- **Name**: `get_device_serial`
- **Description**: This function retrieves the Serial Number of a specified device. It utilizes `udevadm` utility to query device properties and then filters out the Serial Number from the queried properties.
- **Globals**: None.
- **Arguments**:
  - `$1`: The device for which the serial number is to be retrieved.
- **Outputs**: The Serial Number of the provided device, or "unknown" if the serial number cannot be determined.
- **Returns**: Nothing.
- **Example usage**: `get_device_serial /dev/sda1`

3. Quality and security recommendations

1. The function does not handle the cases where the `udevadm` command is not installed or accessible on the system. It's recommended to add a check before executing the `udevadm` command.
2. Tot prevent running with improper arguments, the function should include an initial check to ensure that the proper argument (device name) is provided.
3. Output should be sanitized to prevent potential command injection or processing of unintended data.
4. Include more comprehensive error handling and output informative error messages for better maintainability and debugging.
5. For maximum security, ensure this function runs with minimum required permissions to avoid potential privilege escalation attacks.

6.6.0.36 `get_device_size`

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

Function signature: 1d4e207d13b80b5424975cd10a3ed7197f78e6832fbbf6662e417abdd6def6d2

Function overview The `get_device_size` function in Bash is designed to determine and return the size of a device (or partition) in a Linux based system. This is achieved utilizing the `lsblk` command, which retrieves information about block devices. Should the command fail to execute, the function returns "unknown".

Technical description

- **Name:** `get_device_size`
- **Description:** This function determines the current size of a block device within a Unix-like system.
- **Globals:** None
- **Arguments:**
  - `$1`: The identifier or path of the device to be queried for its size.
- **Outputs:** The size of the queried device, or "unknown" if the device does not exist or cannot be accessed.
- **Returns:**
  - Size of the Device: If the queried device exists and can be accessed.
  - "unknown": If the queried device doesn't exist or can't be accessed.
- **Example usage:**

```bash
### Get the size of /dev/sda device
device_size=$(get_device_size /dev/sda)
echo "Device size is: $device_size"
```

Quality and security recommendations

1. Error Handling: To enhance reliability, incorporate error capturing and handling mechanisms. While "unknown" serves as a failure response, more explicit messaging could improve debugging and user experience.
2. Information Leakage: Avoid exposing detailed system information via error messages which could be utilized by potential attackers.
3. Validate Arguments: Before the function execution, validate if the supplied argument ($1) is a valid device identifier or path.
4. Permission Check: The function should check if it has the necessary permissions to access the device before attempting to query its size.
5. Check Dependency: Ensure `lsblk` command is available in the system environment where the script is intended to run.

6.6.0.37 `get_device_speed`

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

Function signature: a3925bc33db6a5252197ab449a78edcdfb4d9117f6e067cedb4b22c2eaea3a3e

Function overview The `get_device_speed()` function in bash is used to estimate the speed of a device defined by the user. It operates by reading data from the device and writing it to null, which avoids creating unnecessary files. The speed is determined by observing the time taken for the diagnostics to be run using the dd command and then extracted using `grep`. If the device cannot be accessed or read for any reason, "N/A" is returned instead.

Technical description

- **Name**: `get_device_speed`
- **Description**: This function retrieves the speed of a device by reading from it and writing to null using the dd command. The speed is then extracted from the output of the dd command using `grep`. If the device specified cannot be accessed for any reason, the function returns "N/A".
- **Globals**: None.
- **Arguments**:
  - `$1: dev`: this represents the path or identifier of the device whose speed is to be estimated.
- **Outputs**: This function outputs the speed of the device in MB/s, or "N/A" if the retrieval was unsuccessful.
- **Returns**: None.
- **Example usage**:

```
get_device_speed "/dev/sda1"
```

This command will output the estimated speed of the device at path "/dev/sda1".

Quality and security recommendations

1. Check if the device exists and is accessible before attempting to read from it. This can prevent unnecessary errors and ensure accurate results.
2. Sanitize user input to prevent code injection or manipulation of the dd command. Never trust user input without validation.
3. Use more precise methods for speed calculation in production settings as using dd might not be the most accurate.
4. Implement error handling to manage failed read attempts from the device and return meaningful error messages.
5. Consider using buffering and limiting the data read from the device to prevent potential damage or excessive resource usage.

6.6.0.38 `get_device_type`

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

Function signature: c04dccee20f8398f1cc00c1f339464f78c0ea09416025a9627dcbd45477ea68d

Function Overview The `get_device_type` function is a bash shell function designed to determine the device type of a specified device. The function accepts a device name as an argument, and uses `udevadm` info query method to get the relevant properties of the device. It then parses the output to extract the ID_TYPE value. If no such value is found, it defaults to assume that the device is a disk.

Technical Description

- **Function name:** `get_device_type`
- **Description:** The function determines the device type of a specified device using the `udevadm` utility. If it fails to retrieve the device type, it assumes the device is a disk.
- **Globals:** None
- **Arguments:**
  - $1: The device name to query (e.g., `/dev/sda`).
- **Outputs:** Returns the device type (`disk`, `cd`, etc.) to stdout.
- **Returns:** If successful in retrieving the device type, the function returns 0. If it fails to retrieve the device type but defaults to assuming it's a disk, the function still returns 0.
- **Example usage:**

```
device_type=$(get_device_type /dev/sda)
echo $device_type
```

Quality and Security Recommendations

1. Validate the device name provided as an argument to ensure it is a properly formatted device name.
2. Provide an option for the caller to specify the default type (instead of "disk" being the default).
3. Implement thorough error handling and logging for common failure scenarios, such as when the `udevadm` command is not found.
4. Consider adding conditional checks to ensure that "udevadm" is installed and accessible on the host system before proceeding with the function execution.
5. Sanitize output from the `udevadm` command to make sure potential harmful strings will not cause unintended shell command execution.

6.6.0.39 `get_device_usage`

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

Function signature: 859cf1265955125053e13b61c6c4bfa14f3715f652ae3ee914eb7e3927870233

Function Overview The bash function `get_device_usage()` is used to determine the usage status of a given device. By employing a series of commands, it retrieves the mountpoints of the device, ignoring any blank entries, before returning the remaining items as a comma-separated string. If no mountpoints exist for the device, the function will return "unused."

Technical Description

- **Name:** `get_device_usage()`
- **Description:** The function retrieves and returns a comma-separated string of mountpoints for a specific device. If no mountpoints are identified, the function returns "unused."
- **Globals**: (none)
- **Arguments**:
  - `$1`: `dev` The device for which the usage status is required.
- **Outputs**: Writes to stdout a comma-separated string of the mountpoints for the specified device or "unused" if no mountpoints are found.
- **Returns**: (none)
- **Example usage**:

  ```
  get_device_usage /dev/sda1
  ```

Quality and Security Recommendations

1. Input Validation: Ensure proper validation and sanitization of the inputted device name to prevent code injection or other related security issues.
2. Error Handling: Implement robust error handling to ensure the script doesn't crash when the specified device doesn't exist or other unforeseen errors occur when calling the `lsblk` command.
3. Documentation: Remember to update this documentation if there are changes to the function for maintainability and readability.
4. Testing: Regularly test this function with a variety of different devices and contexts to ensure its versatility and reliability.

6.6.0.40 `get_device_vendor`

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

Function signature: e635f4842a790d321a2d2bf3371ad26f62f2881ced6effdff7eaea8887f48cf1

Function Overview `get_device_vendor()` is a Bash shell function that retrieves the name of the vendor for a specified device in a Linux system. It does so by reading a specific file within the system's file structure. If it fails to retrieve this information, it outputs "unknown".

Technical Description

- **Name**: `get_device_vendor()`
- **Description**: This function retrieves the vendor's name of a given device within the Linux system. It reads from the `vendor` file in the `/sys/block/{device}/device/` directory. If it fails to find this file or read from it, it outputs "unknown".
- **Globals**: None
- **Arguments**:
  - `$1`: `dev`. The device identifier.
- **Outputs**: The vendor's name of the device, or "unknown" in case of errors.
- **Returns**: This function does not return a value. It only outputs results.
- **Example Usage**: `get_device_vendor "/dev/sda"`

Quality and Security Recommendations

1. Validate the input: Ensure that the argument passed is a valid device identifier. This can be accomplished by using a regular expression for validation.
2. Manage errors effectively: Instead of silently returning "unknown" when something doesn't work, it could be helpful to return different error messages depending on the encountered error (file not found, permission denied, etc.). This would give more insight to the users of the function.
3. Consider permissions: As this function requires reading files that might require special permissions, ensure that this function is run with the correct permissions, or handle the potential permission errors elegantly.
4. Protect against Command Injection: Although this function uses `basename` to only take the last part of the device path, it would be safer to ensure that no characters that can alter the function behaviour are accepted as input.
5. Properly handle spaces in vendor names: The function currently removes all spaces from vendor names. Any vendor name with a space will thus be changed. If this behavior is not desired, the `tr -d ' '` part should be removed. Conversely, if it is known that certain vendors have undesired trailing or leading spaces in their names, only these should be targeted.

6.6.0.41 `get_external_package_to_repo`

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

Function signature: a00cf7324f6795dbbcefc882d9a0346242d31cdeaef0ed00e85c79720244ebc9

Function Overview The `get_external_package_to_repo` function is a shell script in Bash that downloads an RPM file from a given URL and then stores the file in a specified repository path. This function utilizes error-checking and exit codes to manage potential issues such as a non-existent repository or a non-RPM file.

Technical Description The function can be defined as follows: - **Name:** `get_external_package_to_repo` - **Description:** Downloads an RPM file from a provided URL and saves it to a specified repo path. - **Globals:** No global variables are modified. - **Arguments:** - `$1`: URL of the RPM file to be downloaded. - `$2`: Path to the target repository where the downloaded file will be stored. - **Outputs:** Log messages indicating the process and any potential errors. - **Returns:** - `0` if the download and save are successful. - `1` if URL and/or repo path are not specified. - `2` if the repo path does not exist. - `3` if the URL does not point to an RPM file. - `4` if the download fails. - **Example Usage:** `get_external_package_to_repo "http://source.com/myfile.rpm" "/path/to/my/repo"`

Quality and Security Recommendations

1. Ensure input validation to prevent potential security vulnerabilities like Path Traversal.
2. Verify network connectivity before trying to download the file to handle potential network failures.
3. Add a checksum validation step after downloading the file to ensure file integrity and authenticity.
4. Implement more robust error handling and logging to facilitate troubleshooting.
5. Introduce permissions check for the target directory to ensure the script has write access before download.

6.6.0.42 `get_host_type_param`

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

Function signature: 2aa086f62cbb99876d8c7312d176dfce1f88af4d006d3f9a64537fa96e3f9008

Function overview The `get_host_type_param()` function in Bash is designed to retrieve a specific parameter from a given host type. This is performed by passing the host type and the parameter key as arguments.

Technical description **Name**: `get_host_type_param`

Description: This function retrieves a specific parameter (key) from a given host type (type). The ‘declare -n’ command is used to create a nameref ‘ref’ which acts as a reference to the variable with the name given by ‘type’. It then echoes the value of the element in ‘ref’ with the index ‘key’.

Globals: None

Arguments: - `$1 (type)`: The name of the host type from which the parameter is to be fetched. - `$2 (key)`: The key of the parameter to be fetched.

Outputs: It prints the value of the desired parameter onto the stdout.

Returns: It does not return anything except for the output printed onto stdout.

Example usage:

```
declare -A serverA=("os"="linux" "ip"="192.168.1.1")
get_host_type_param serverA "os"  # this will output "linux"
```

Quality and security recommendations

1. Avoid using the function’s arguments directly in the ‘declare -n’ command. Instead, validate them first. Bash doesn’t prevent the creation of namerefs to readonly variables, which may lead to unexpected behavior.
2. For security reasons and to ensure correct fetching of indexed array elements, always quote the keys when using them to access associative arrays.
3. Make sure to error-check the existence of the array and the specific key before trying to access it, to prevent uncontrolled output.
4. Consider applying a variable naming convention in order to prevent conflicts between global and local variables which might cause erroneous behavior.
5. Lastly, remember to encapsulate the function body with curly braces `{}` for readability and maintainability.

6.6.0.43 `get_iso_path`

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

Function signature: c9a43a62bf1aba8fb474972e5a14edb51e00a6b3ad7b99ced97edcfedfb00929

Function overview The `get_iso_path` function is a bash shell function that gets the ISO file path from a specific directory defined by the variable `HPS_DISTROS_DIR`. It checks whether the `HPS_DISTROS_DIR` variable is set and points to a valid directory. If the condition is met, it concatenates `/iso` to the directory path and outputs this new path. If the condition is not met, it outputs an error message and returns a non-zero exit status to indicate the error.

Technical description

- **Name**: `get_iso_path`
- **Description**: This function checks if the `HPS_DISTROS_DIR` variable is defined and represents a valid directory. If so, it creates a new path by appending `/iso` to the directory path. If not, it sends an error message to standard error and returns an exit status of 1.
- **Globals**:
  - `HPS_DISTROS_DIR`: The path to the directory where the ISO file is located.
- **Arguments**: None.
- **Outputs**:
  - If successful, returns the path of ISO file.
  - If unsuccessful, returns an error message in stderr.
- **Returns**:
  - Returns 0 if successful (standard for bash functions).
  - Returns 1 if unsuccessful.
- **Example usage**:

```
$ get_iso_path
```

Quality and security recommendations

1. When dealing with global variables, check their status before using them. A global variable like `HPS_DISTROS_DIR` should be verified for validity not only within this function but also where it is set initially.
2. Error messages should be meaningful and guide the user to resolve the issue. "[x] HPS_DISTROS_DIR is not set or not a directory." is a clear and informative error message.
3. Shell scripts can be risky if not properly handled. Always validate and sanitize the inputs, ensure proper permissions are set, and handle errors gracefully to prevent security vulnerabilities.
4. Adding comments to the function to explain its operation would improve the understandability of the code.

6.6.0.44 `get_provisioning_node`

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

Function signature: 1bbd682413f1c35f8f147b598c84f60028eea1205d56405e50b38d7d551b5b01

Function overview The function `get_provisioning_node` is responsible for grabbing and returning the default gateway IP address which corresponds to the provisioning node in a network. It uses the `ip route` command in conjunction with awk for parsing the output properly and retrieving the pertinent information.

Technical description **Name**: `get_provisioning_node`

Description: This function retrieves and prints the IP address of the default gateway in the network, often called the provisioning node. It is commonly used in networking and system administration scripts for setting up or troubleshooting networks.

Globals: None.

Arguments: None.

Outputs: The default gateway IP address.

Returns: Prints to stdout the provisioning node (default gateway IP address), or nothing if the command does not find a default gateway.

Example usage:

```
$ get_provisioning_node
192.168.1.1
```

In this example, `192.168.1.1` is the IP of the provisioning node.

Quality and security recommendations

1. Error Handling: Add error checking to ensure `ip route` command is successful. If the command fails, the function should handle the error and not continue to parse output.
2. Permissions: Ensure the script running this function has appropriate permissions. Command like `ip route` may require higher privileges, it's necessary to control who and how can run the script.
3. Dependency Checks: Check if awk is available on the system. This command is critical for the function to work as expected.
4. Privacy Protection: Be cautious about where you output the IP addresses as they may be sensitive information in certain contexts.
5. Code Reusability: This function can be part of a larger library of networking functions to increase reusability and maintainability.

6.6.0.45 `handle_menu_item`

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

Function signature: efc740d42cbef4cac8830b693cf786a6fe836fdba9b80db8e8eef1b0651d2aa6

Function overview This bash function, `handle_menu_item()`, acts as a comprehensive handler for various menu functions across different menus within a system. It accepts two arguments: a menu item and a media access control (MAC) address, and then performs certain operations based on the specific menu item chosen.

Technical description

- **Name**: `handle_menu_item()`
- **Description**: The function serves to handle various menu items across different menus within a system. Its behavior varies depending on the passed menu item.
- **Globals**: N/A
- **Arguments**:
  - `$1 (item)`: The menu item that must be handled.
  - `$2 (mac)`: The MAC address associated with the menu item.
- **Outputs**: The function can output log messages, error messages or other strings based on the menu item passed.
- **Returns**: The function does not explicitly return a value but it might exit out or break the program in certain conditions (for instance, if an unknown item is passed).
- **Example usage**: `handle_menu_item 'init_menu' '00:1B:44:11:3A:B7'`

Quality and security recommendations

1. Consider validating the inputs (menu item and MAC address) to mitigate unintended behaviors or vulnerabilities in the function.
2. It might be beneficial to define all possible options for the menu item in a list or another manageable construct to enhance the readability and maintainability of the function.
3. Make sure the function components like `ipxe_init`, `ipxe_install_hosts_menu`, etc., that the function depends on, are secure and error tolerant themselves.
4. Consider thoroughly testing this function with different inputs as it seems to have a broad impact on the overall system based on the passed menu item.
5. As part of secure coding practices, ensure that logging information does not reveal any sensitive or unencrypted system data.

6.6.0.46 `has_sch_host`

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

Function signature: 379b6704d9af414555293ef691b70449d39306ef567e44aabf9ef70f91fa692d

Function Overview The function `has_sch_host()` is designed to ascertain if there exists at least one host with a 'SCH' type within the specified host configuration directory. In case the directory is not present, the function outputs an error message and terminates with an exit status of 1. If the host of 'SCH' type is found, the function returns an exit status of 0 indicating a successful find. Conversely, if no such host entry is found, the function returns an exit status of 1, indicating failure to find any 'SCH' type hosts.

Technical Description **Name:** `has_sch_host`

Description: This function checks for at least one host with 'SCH' type in a host configuration directory.

Globals:

- `HPS_HOST_CONFIG_DIR`: This global variable represents the path to the host configuration directory.

Arguments: No argument is expected for this function.

Outputs: If the host configuration directory is not available, the function will output an error message: "[x] Host config directory not found: <path_to_the_directory>"

Returns: The function returns the following values based upon its operations: * `0`, if at least one host with 'SCH' type is found within the host configuration directory. * `1`, if the host configuration directory could not be found or if no such 'SCH' type hosts are found.

Example Usage:

```
source script_containing_has_sch_host
HPS_HOST_CONFIG_DIR="/path/to/host/config/dir"
has_sch_host

### It will return 0 if 'SCH' type host is found or 1 otherwise.
```

Quality and Security Recommendations

1. Use of `local` for variable scope: Local scope is great, but this could be elevated by examining whether the use of `local` here is appropriate or if `declare` might better serve the purpose to prevent variable masking.
2. Explicit variable naming: The variable name `host_dir` might benefit from being a bit more descriptive, e.g., `targetHostDir`.
3. Error handling: Pay attention to error handling for the `grep` command as well. There could be situations where the `grep` command might fail.
4. Security: If `HPS_HOST_CONFIG_DIR` can be controlled by an external user, it's theoretically possible to craft a path that leads to unsafe areas. Examine and sanitize if needed.
5. Efficiency: The script uses `grep`, which is fine, but to improve speed, consider using built-in shell scripting commands or tools, which are usually faster than spawning a new process to run `grep`.

6.6.0.47 `host_config`

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

Function signature: a3f0f5a97e6c9b8ed42cd66bea9ba663ff8f3a40bc906925763eb0926ef1c253

Function overview The `host_config` function is a bash function specifically designed to read, write, update, or query entries on a local configuration file identified by a MAC address. It systematically reads a configuration file and stores its key-value pairings in an associative array. After reading the file, the function is able to execute a number of commands ("get", "exists", "equals", "set") based on the arguments provided, on the associative array to manage the configuration entries. Should the command be unknown or absent, the function will return an error.

Technical description

- **Name**: `host_config`
- **Description**: A shell function for managing the configuration of a host identified by its MAC address. It interactively reads, checks, compares, and updates configuration data stored on a local configuration file.
- **Globals**: [VAR: __HOST_CONFIG_PARSED: (boolean) Flag indicating if the Host Configuration File has been parsed, HOST_CONFIG_FILE: the location of the host configuration file, HPS_HOST_CONFIG_DIR: the directory containing the host configuration file, HOST_CONFIG: (assoc array) Variables parsed from the Host Configuration File.]
- **Arguments**:
  - $1: `mac` — the MAC address associated with the host configuration file.
  - $2: `cmd` — the command to execute on the host configuration ("get", "exists", "equals", "set").
  - $3: `key` — the key from the configuration to operate on.
  - $4: `value` — the value to compare or set for the provided key.
- **Outputs**: Depending on the cmd argument, it can either:
  - print the value corresponding to the given `key`
  - print a bool indicating if the `key` exists or if the `key` equals the `value`
  - print logs indicating performed updates
  - print an error message in case of invalid `cmd`
- **Returns**: It could return nothing, or return error code 2 in case of invalid `cmd`
- **Example usage**:

```
host_config 00:0a:95:9d:68:16 set VAR Test
```

Quality and security recommendations

1. Always validate inputs: To ensure the function handles only intended input, add checks to validate the format of the MAC address, and that the `cmd`, `key`, and `value` inputs are not malicious or invalid.

2. Incorporate error handling for file operations: Implement necessary checks regarding file availability, read and write permissions for the config file, and handle errors that might occur during file operations.
3. Add a function documentation: Include a comment block at the beginning of the function to explain its purpose, parameters, return values, and sample usage.
4. Implement detailed logging: Integrate with a logging system to trace every operation performed by this function in a systematic manner. This could greatly simplify debugging and auditing.
5. Check for command success: Add checks to ensure that commands completed successfully and handle any errors that may occur.
6. Use more secure methods for sensitive data: If the configuration data is sensitive, consider using more secure methods to store and handle it, such as encryption and secure file permissions.

6.6.0.48 `host_config_delete`

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

Function signature: e05b6467ee0958047fe16e2eb5a0519a4ceab375f01a2f8a27e8ee6f35cf7400

Function Overview The `host_config_delete()` function is primarily used to delete the configuration file associated with a specific host identified by its MAC address. It constructs the configuration file path using the provided MAC address and a global variable representing the directory path. If the file exists, it is deleted and an informational log message is generated, otherwise, a warning message is logged.

Technical Description

- **Name**: `host_config_delete`
- **Description**: Deletes a host configuration file associated with a provided MAC address.
- **Globals**:
  - `HPS_HOST_CONFIG_DIR`: Represents the directory path where host configuration files are present.
- **Arguments**:
  - `$1`: Represents the MAC address of the host machine, which is also used as the config file name.
- **Outputs**: Logs an informational message when the configuration file is deleted successfully, else logs a warning message if it fails to locate the file.
- **Returns**: The function returns `0` when the configuration file is deleted successfully and `1` when the file is not found.
- **Example usage**:

```
host_config_delete "fa:16:3e:d7:f2:6c"
```

Quality and Security Recommendations

1. Always validate inputs: The function should validate the provided MAC address format before proceeding.
2. Sanitize file paths: Consider edge cases where the value of `HPS_HOST_CONFIG_DIR` could lead to unwanted behavior (e.g., directory traversal attacks).
3. Use `-v` option to run the function in verbose mode: It helps in debugging if any issue arises.
4. Error messages should be specific: This helps in understanding the problem faster in case of errors.
5. Confirm before deleting files: The function currently deletes the file without any confirmation. A prompt for confirmation before deleting any file would be a safer method.

6.6.0.49 `host_config_exists`

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

Function signature: 93698d3265775cdeb0581fb007522cdb74832e2c705812027e67b512cd33964b

Function overview The `host_config_exists` function in Bash is used to check if a host configuration file for a specific MAC address exists. It does this by constructing the file path using a globally defined configuration directory and the MAC address (passed as an argument), and then checking if a file at that path exists.

Technical description

```
- Name:   host_config_exists

- Description:  This function checks for the existence of a host
 ↪  configuration file for a certain MAC address within a
 ↪  predetermined host configuration directory.

- Globals:
    - HPS_HOST_CONFIG_DIR: The directory where host configuration
     ↪  files are stored.

- Arguments:
    - $1: The MAC address of the host whose config file is being
     ↪  searched for.

- Outputs:  Outputs nothing to stdout or stderr, though Bash will
 ↪  naturally output an error message to stderr if there's an
 ↪  unexpected problem (e.g. insufficient permissions to access
 ↪  the directory).
```

```
- Returns:  Returns with 0 status if the host config file for the
↪  provided MAC address exists, otherwise returns with 1 status.

- Example Usage:
```

host_config_exists "00:11:22:33:44:55"

```
If there's a config file "00:11:22:33:44:55.conf" in the directory specified by `${HPS
```

Quality and security recommendations

1. Make sure that the HPS_HOST_CONFIG_DIR variable is set to a secure directory that only trusted users have read and write access to. This is to prevent any unauthorized access or changes to configuration files.
2. Validate the MAC address input in the function. This could be done by checking the format of the input to ensure that it follows the MAC format 6 groups of two hexadecimal digits.
3. Check that the discerned config file path is within the expected bounds to avoid any potential for directory traversal.
4. Avoid using relative paths for the HPS_HOST_CONFIG_DIR to ensure that the function's output is always consistent regardless of the current working directory.

6.6.0.50 `host_config_show`

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

Function signature: 6105ece190686264b3985f4c2de384ff48edc977456deeb1f70f8a863bb065a8

Function Overview The function `host_config_show()` is designed to read and display a configuration file for a host in a network. Accepting a MAC address as a parameter, it looks for the corresponding configuration file in the `HPS_HOST_CONFIG_DIR` directory, which is defined elsewhere in the script. If the configuration file exists, the function reads it through line by line. For each line it extracts a key and a value separated by =. If any value exists for the key, then it's presented in a specific format. Also, the function logs an informational message if no configuration file exists for the given MAC address.

Technical Description

- **Name:** `host_config_show()`
- **Description:** Reads and displays a host's configuration file based on its MAC address.
- **Globals:** `[ HPS_HOST_CONFIG_DIR: Directory where host configuration files are stored.]`

- **Arguments:** `[ $1: The MAC address of a host in the network. ]`
- **Outputs:** Outputs each key-value pair from the config file, with special characters such as quotes and backslashes in values properly escaped. If no configuration file exists for the provided MAC address, it logs an informational message.
- **Returns:** Returns `0` if no configuration file exists, signaling that the function ran successfully with this outcome.
- **Example Usage:**

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

Quality and Security Recommendations

1. Sanitize the input to ensure that the MAC address is in a valid format. This can help to prevent potential command injection attacks or unintended behavior.
2. Check if the `HPS_HOST_CONFIG_DIR` global is set before attempting to use it. If it's not set, the function should return an error.
3. For a more resilient design, handle unexpected errors such as reading from a corrupt config file or issues with file permissions.
4. Avoid disclosing too much information in log files to prevent potential information leakage.
5. Log not only the absence of a configuration file, but also when a file is found and successfully processed.

6.6.0.51 `host_initialise_config`

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

Function signature: 9cbdb13ea8edf79033b1a96edf52ebe54e082910b6a8a403b6b2fd86a4d5b486

Function Overview The `host_initialise_config` function is designed to initialise a configuration file for a host using the MAC address. The function first ensures that a directory for host configurations exists, and then sets the state of the host configuration to "UNCONFIGURED". Afterwards, the function logs the initialisation of the host configuration.

Technical Description Here is the technical description for `host_initialise_config`:

- **name**: `host_initialise_config`
- **description**: Initialises a configuration file for a host using the MAC address. The state of the host configuration is set to "UNCONFIGURED", and the initialisation is logged. The function should be invoked with the MAC address as an argument.
- **globals:**

  - HPS_HOST_CONFIG_DIR: The directory in which the host configuration files are stored
- **arguments:**
  - $1: MAC address of the host for which the configuration file must be initialised
- **outputs**: Logs the initialisation of the host configuration file
- **returns**: None
- **example usage**: `host_initialise_config "00:0a:95:9d:68:16"`

Quality and Security Recommendations

1. Make sure to use a valid MAC address as the argument when calling `host_initialise_config`.
2. Verify that the `${HPS_HOST_CONFIG_DIR}` directory exists or the script has the necessary permissions to create it. This is to avoid failures in `mkdir`.
3. Commented out parts of the code (i.e., lines for adding timestamps and echoing the configuration file) should be removed or uncommented if they are needed. Leaving large parts of code commented out can lead to confusion.
4. Validate the input to prevent potential security vulnerabilities.
5. Handle potential errors in the function, like issues while creating directories, setting states, or logging information.

6.6.0.52 `host_network_configure`

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

Function signature: 2d391e7910d04552e8ca1284295323649c38c44df9d4c9f8e335c080e4d38dac

Function overview The `host_network_configure` function is responsible for facilitating network configuration on a host machine. It leverages identifiers such as the network macid and hosttype alongside existing network parameters such as the DHCP_IP and DHCP_CIDR retrieved from the cluster configuration. The function also checks if `ipcalc` is installed and then gets the netmask and network base using the `ipcalc` tool to further utilize in network configuration. It immediately exits with an error status if any of the needed configurations are missing or if the `ipcalc` tool is absent.

Technical description

Name `host_network_configure`

Description This function is used to configure the network settings of a host machine in a cluster.

Globals

- DHCP_IP: IP address in the DHCP configuration
- DHCP_CIDR: CIDR block details in the DHCP configuration

Arguments

- `$1`: MAC ID of the host need to be configured
- `$2`: The type of the host, such as master or worker

Outputs This function does not produce any explicit output. It exerts its effect by changing the network configuration of the host.

Returns This function will return `1` when certain necessary configurations are not found or the `ipcalc` command is not present on the host machine.

Example usage `host_network_configure "mac-address" "host-type"`

Quality and security recommendations

1. Consider adding validation steps for MAC ID and host type.
2. Ensure that sensitive information in logs such as MAC ID or IP addresses is protected and not exposed.
3. `ipcalc` dependency should be checked early in installation or startup scripts to reduce the runtime failures.
4. Explicit error handling can be introduced for potential errors during command execution.
5. More robust return codes should be used to indicate different error scenarios.

6.6.0.53 `hps_log`

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

Function signature: e07a1142dabfc3ddeb37abc77adca2f2b9ddd9913a41b1be69c52f6bf83d7303

Function Overview The `hps_log` function is designed to log events or messages in a Linux/Unix environment. This function allows for logging of multiple levels and uses a default identifier and log directory if none are provided. Events are time-stamped upon logging.

Technical Description

- **Name**: hps_log
- **Description**: A logging function in Bash programming. It can take any number of arguments, with the first one being the level of the message and the rest of them being the message. It uses the ident and logdir predefined global variables if they are set, or their default values if they are not set.
- **Globals**:
  - HPS_LOG_IDENT: Defines the identifier for the log (default is "hps").
  - HPS_LOG_DIR: Defines the directory for the log file (default is "/var/log").
- **Arguments**:
  - $1: Logging level, e.g., error, warning etc.
  - $2: Message to be logged.
- **Outputs**: Logged messages with a timestamp, level designation, and identifier which are written to a specified log file.
- **Returns**: This function does not return any value.
- **Example Usage**:

```
hps_log "error" "This is an error message"
```

Quality and Security Recommendations

1. Check variable contents: Always check the contents of variables before passing them to this function to avoid any form of command injection or formatting issues.
2. Regular cleanup: Set up a regular cleanup process or log rotation for your log files to avoid them consuming too much disk space.
3. Secure log files: Make sure that the permissions on your log file directory and the log files themselves are set to allow only authorized users to read or modify them. This helps to prevent unauthorized access or modifications.
4. Handle errors: Consider improving this function by including error handling features, such as what should happen if the log file cannot be written to.

6.6.0.54 `hps_services_restart`

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

Function signature: df81ff0c0a6321687bffa9f72094b354899875b4a86b70f93f1a5eaa47f47642

1. Function overview The `hps_services_restart()` function is designed to restart all services under supervision. This function achieves this by first configuring supervisor services using the `configure_supervisor_services` function. Then, it reloads the supervisor configuration by way of the `reload_supervisor_config` function. Finally, it uses the `supervisorctl` command to restart all services, passing the path to the supervisord.conf configuration file via the `HPS_SERVICE_CONFIG_DIR` environment variable.

2. Technical description

- **name**: `hps_services_restart`
- **description**: This function is used for restarting all services managed by supervisor. It first sets up supervisor services, reloads the configuration, and then restarts all services.
- **globals**: [`HPS_SERVICE_CONFIG_DIR`: It is a global variable that points to the directory holding the Supervisor services configuration file]
- **arguments**: None.
- **outputs**: Initializes and restarts all supervisor services. Any output (such as error messages) would come from the internals of these commands, sent either to stdout or stderr.
- **returns**: Return status is dependent on the underlying commands within the function which do not have any specific return value checks or handling mechanism.
- **example usage**: The function can be called directly in the script where it's defined as `hps_services_restart`

3. Quality and security recommendations

1. This function directly restarts all services without performing any checks. It would be better to implement a mechanism to first check the status of the services and then restart only those which are not running properly.
2. Handle possible errors or exceptions from the `configure_supervisor_services`, `reload_supervisor_config`, and `supervisorctl` commands to improve reliability.
3. Avoid use of globals where possible as it might introduce side effects. Prefer to use specific, function-scoped variables.
4. Ensure appropriate permissions and user-level privileges when working with system services. This includes ensuring the right user context when calling `supervisorctl` or other service management commands.
5. Include proper logging mechanisms to track the function execution process for better troubleshooting.
6. Implement return checks or handling mechanisms to manage the function's return status accordingly.

6.6.0.55 `hps_services_start`

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

Function signature: d803223c5cd9a326d513c4b57b5085266767da7cb4e9c6c99acebea677274834

Function Overview The function `hps_services_start` essentially manages the starting of services in a given system configuration. It does so in a three-step process - it first configures the supervisor services, then reloads the supervisor configuration, and finally initiates all services as specified in the supervisor configuration.

Technical Description

- **Name**: `hps_services_start`
- **Description**: This function starts all services as defined within a specified supervisor configuration. It's a part of the system initialization and process management which includes supervisor configuration load, refresh, and service starting.
- **Globals**:
  - `HPS_SERVICE_CONFIG_DIR`: Indicates the directory where the supervisor configuration file is stored.
- **Arguments**: No arguments required.
- **Outputs**: Starts services based on the supervisor configuration file located in `HPS_SERVICE_CONFIG_DIR`.
- **Returns**: None.
- **Example usage**: `hps_services_start` - it is generally used without arguments.

Quality and Security Recommendations

1. Provide error handling for situations where the config file is missing from the `HPS_SERVICE_CONFIG_DIR`.
2. Ensure appropriate permissions for the `HPS_SERVICE_CONFIG_DIR` and the configuration file `supervisord.conf` - it should not be editable/deletable by unauthorized users.
3. To prevent possible service interruptions, add a check verifying if the services started successfully after the execution of `hps_services_start`.
4. Ensure that supervisor services are appropriately configured before trying to start them.
5. Consider informing the user about the status of services after they are started.
6. Follow a strict naming convention for services in the supervisor configuration file.
7. Make sure the function is covered with unit tests that emulate different environments and scenarios.
8. Use an encrypted connection when communicating with the supervisord service to prevent any security breaches.
9. Avoid using shell execution (`supervisorctl -c`) where possible as it poses a security risk due to possible command-injection attacks.

6.6.0.56 `hps_services_stop`

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

Function signature: 7353ea08d49309b1aa8e3187b443507d430fd727f57e8ecc29accbe8383b6169

Function overview The `hps_services_stop` function is a Bash function designed to stop all services being managed by Supervisor. Supervisor is a client/server system that allows its users to monitor and control a number of processes on UNIX-like operating systems. `supervisorctl` is the command-line client piece of the supervisor. It provides a shell-like interface to the features provided by the Supervisor server. This function can be particularly useful in scripts or as a part of larger system operations where certain services need to be stopped at certain times.

Technical description

- **Name**: `hps_services_stop`
- **Description**: This function stops all services that are being managed by `supervisorctl` by explicitly calling the `stop all` command on `supervisorctl`. The function utilizes a configuration directory specified by the `HPS_SERVICE_CONFIG_DIR` global.
- **Globals**: [`HPS_SERVICE_CONFIG_DIR`: The directory where the `supervisord.conf` file is located. This config file is used by `supervisorctl` to control the services.]
- **Arguments**: None are required for the function to properly execute.
- **Outputs**: Will output any information or errors generated by the `supervisorctl stop all` command.
- **Returns**: Returns the exit status of the `supervisorctl stop all` command.
- **Example usage**: `bash hps_services_stop`

Quality and security recommendations

1. Be cautious with who has access to running this function, as stopping all services can have dramatic implications.
2. Ensure that there is proper error handling both within the function and wherever the function is being called from. Always check the return status of critical function calls.
3. Find ways to limit the usage of global variables as they can create hard dependencies across your code, which may lead to unexpected behavior and hard to debug code.
4. For added security, you may want to consider implementing additional checks to ensure that the `supervisord.conf` file is in the correct location and has the correct permissions before attempting to run the `supervisorctl` command.

6.6.0.57 `initialise_cluster`

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

Function signature: 0ec03980bf3af34dd55fa2d767b0fc494f13846b6935087299fd9161e89ab908

Function Overview The `initialise_cluster` is a Bash function that serves the purpose of cluster initialization. This function takes a cluster name as input, builds a cluster directory path using base directory and cluster name, and creates a cluster configuration file. If no cluster name is supplied, or if the cluster directory already exists, the function halts execution and returns an error. If the initialization process is successful, the function prints a success message to the console, and then calls the `export_dynamic_paths` function to dynamically export cluster paths.

Technical Description **Name:** `initialise_cluster`

Description: Initializes a new cluster by creating its directory and configuration file based on the supplied cluster name.

Globals: `HPS_CLUSTER_CONFIG_BASE_DIR`: The path to the base directory containing the cluster configuration.

Arguments: `$1 - cluster_name`: The name of the cluster to be initialized.

Outputs: - An error message to STDERR if no cluster name is provided or if the cluster directory already exists. - A message indicating the successful initialization and configuration of the cluster.

Returns: - 1 if no cluster name was provided. - 2 if the cluster directory already exists. - 3 if there was an error in exporting cluster paths. - If the function is successful, it does not explicitly return a value.

Example Usage:

```
initialise_cluster "my_cluster"
```

Quality and Security Recommendations

1. After the cluster directory and file are created, it would be recommendable to set the appropriate permissions to secure these.
2. Always validate user-supplied input, such as the cluster name, to prevent any possible exploits like directory traversal.
3. It's crucial to have error handling at every function call that could potentially fail, like `mkdir` and `cat`.
4. For better readability and maintainability, consider using longer, more descriptive variable names.
5. Make sure to document the globals and their expected values or defaults. The `HPS_CLUSTER_CONFIG_BASE_DIR` global is used without any prior default value setting or declaration, which might lead to unexpected behaviour if it's not set in the environment.
6. Be consistent with error reporting. If the function fails at any point, it should be clear to the caller what caused the failure.

6.6.0.58 `initialise_distro_string`

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

Function signature: c440af9a86acddde30570b73ae6d52c7bddf38d765f513ea26ebf15ee4805200

Function Overview The function `initialise_distro_string` uses local variables and information taken directly from the operating system to populate a string in a specific format. This string includes the architecture of the machine, the manufacturer, the name of the operating system and its version.

Technical Description **Name:**
`initialise_distro_string`

Description:
This Bash function is used to identify specific details about the host machine's operating system, including architecture (via `uname -m`), manufacturer (defaulted to 'linux') and, if available, the operating system's name and version (via `/etc/os-release`). If the OS name and version cannot be determined, they are set to 'unknown'. The function returns a string comprising these elements, formatted as: `cpu-mfr-osname-osver`.

Globals:
No global variables used.

Arguments:
No arguments are required.

Outputs:
Produces a string formatted as follows: `cpu-mfr-osname-osver`

Returns:
The final string combining the cpu, manufacturer, OS name and version.

Example Usage:

```
distro_string=$(initialise_distro_string)
echo $distro_string
```

Quality and Security Recommendations

1. It is advised to add validation of the inputs before using them.
2. Handle the case when the `/etc/os-release` file does not exist or is inaccessible. Right now, the function defaults to 'unknown' in such a case, but perhaps a warning message, error code or a fallback mechanism could be implemented.
3. A division of the function into smaller, more specific functions could be considered to enhance readability and maintainability.
4. The function relies on default variables set by external systems like OS. This could be secured by sanitizing these variables before using them in case these external systems get compromised.
5. Test this function across different operating systems or variations of Linux to ensure it works reliably across all.

6. Error handling should be added to ensure the function behaves as expected even in unforeseen situations.

6.6.0.59 `initialise_host_scripts`

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

Function signature: 77a8633d9660b15df9cec14573030ab29237d1a0215a723d9e4bd1c13bb45d38

Function Overview The `initialise_host_scripts()` function in Bash scripting performs a series of operations. It starts by declaring and initializing two local variables, `gateway` and `distro`, which are used to acquire respective information from the `get_provisioning_node` and `initialise_distro_string` functions. A URL is then constructed utilizing the said variables along with the pre-defined URL structure and stored in `url`. The script then retrieves the script from the URL and places it in a specified destination (`dest`). If the script is successfully fetched, it is sourced; otherwise, it echoes an error message and returns 1.

Technical Description

- **Name:** `initialise_host_scripts`
- **Description:** Retrieves a bundle of functions from a specified URL and sources it.
- **Globals:** None
- **Arguments:** No arguments are required.
- **Outputs:**
  - Script fetching status
  - Error message in case the fetching fails
- **Returns:**
  - Does not return anything in a successful run.
  - Returns `1` when the script fails to fetch the host functions.
- **Example usage:** `initialise_host_scripts` No arguments necessary to call this function.

Quality and Security Recommendations

1. Incorporate input and output validation: This is a critical practice that assists in mitigating security risks associated with tainted or illegitimate data.
2. Use HTTPS protocol for URL: Using HTTP can expose the script to potential man-in-the-middle attacks. HTTPS should be the default to ensure secure transmission.
3. Add more error checks: Perform checks on the status of the gateway and distro variables and factor in additional handling for potential errors.
4. Enhance logging: To further provide detailed context-specific error and status messages for better debugging, increase logging verbosity.

5. Define a timeout for the `curl` operation: This will prevent the script from hanging indefinitely in cases where the specified URL is facing issues.
6. Avoid global variables: The function does not presently use any, and this should continue to prevent potential conflicts and bugs.

6.6.0.60 `int_to_ip`

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

Function signature: 6dbd1e706292fd976250f7700867b4102392c8a8b5c39d6e0ac9e7db0bc9233c

Function Overview The `int_to_ip` function is a bash function that converts an integer into an IP address format using bitwise shift and bitwise AND operations. It is used in the lower part of the function to convert network, broadcast, gateway IP addresses, and to calculate the start and end of an IP range.

Technical Description

- **Name**: `int_to_ip`
- **Description**: This function is used to convert an integer to an IPv4 address. It takes an integer as an argument and calculates each octet of the IP address by shifting the bits of the input integer to the right and then applying bitwise AND operator with 255 to get the value of the octet.
- **Globals**
  - VAR: Not applicable
- **Arguments**
  - `$1`: This argument is the integer to be converted into an IP address.
- **Outputs**: This function outputs the equivalent IP address of the input integer.
- **Returns**: Not applicable
- **Example Usage**:

  ```
  local ip_integer=2130706433
  int_to_ip $ip_integer
  # Output: 127.0.0.1
  ```

1. Ensure to validate the input to the `int_to_ip` function. The function currently assumes valid integer inputs that can be converted into an IPv4 address.
2. Update the function to handle non-integer inputs gracefully. Currently, the function does not check if the input is indeed an integer and can behave unpredictably with non-integer values.

3. Add a return statement in the function to indicate a successful process or failure, which can be useful in error-handling. Currently, the function does not return any status.
4. Improve comments and add inline documentation to enhance code readability and ease future updates or bug-fixing initiatives. The existing comments could be expanded to detail more about what the code is doing.
5. Carefully manage and control access to your scripts to prevent unauthorized viewing or modification, thus enhancing the security of your application.

6.6.0.61 `ip_to_int`

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

Function signature: 5857fba825712df3f4e604f317140e0bfcae4ede9a084dd6ca01ab8ec446973f

Function overview This Bash function `ip_to_int()` is capable of converting an IPv4 address into a corresponding integer. The function uses Internet standard dotted-decimal IPv4 addresses (e.g., 192.0.2.0), splits the IP at its dots, then calculates and prints out the integer representation of the IP.

Technical description

- **Name**: `ip_to_int`
- **Description**: This Bash function inputs an IPv4 address as an argument and converts it into its corresponding integer value using bit shifting and arithmetic expression. The result is returned to STDOUT.
- **Globals**: None
- **Arguments**: [`$1`: IP address to be converted in the format of x.x.x.x where 'x' are numeric values ranging from 0 through 255.]
- **Outputs**: The integer value representing the inputted IP address
- **Returns**: Null
- **Example usage**: `ip_to_int "192.0.2.0"` will output 3221225984.

Quality and Security Recommendations

1. For safer code, input validations could be carried out, for instance, checking if the IP address is valid before further processing.
2. Consider checking the number of input arguments to ensure the script works with correct data. If no argument (or more than one) is given, the function should inform the user and exit with a non-zero status.

3. Despite the fact that bash handles integer overflows by wrapping around, the function might be subjected to IP addresses whose integer representations are outside the bounds of what can be expressed in 32 bits. It would be safer to monitor or limit this.
4. The function should handle possible interruptions and should be able to clean up or leave a comprehensible state of execution if interrupted.
5. You could potentially improve readability by using more descriptive variable names rather than o1, o2 etc.
6. Although not a security issue, the function could be more flexible by accepting IP addresses in formats other than one single string argument (for instance accepting 4 separate byte arguments).

6.6.0.62 `ipxe_boot_from_disk`

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

Function signature: f26dc79cd8c9ea270e2758702b63d4a607fd3fd40737c5d947de898af37ff1fe

Function overview The function `ipxe_boot_from_disk` is designed to boot from a local disk through the bios by terminating the iPXE stack. This function is part of the iPXE pre-boot execution environment system, which allows network boot from a client system. It functions by displaying a message ("Handing back to BIOS to boot") and then delays for 5 seconds prior to exiting the function, effectively leading to the BIOS booting.

Technical description

- **Name:** `ipxe_boot_from_disk`
- **Description:** This is a bash function designed to boot from the local disk via BIOS with iPXE stack termination.
- **Globals:** None.
- **Arguments:** This function does not take any arguments.
- **Outputs:** A message "Handing back to BIOS to boot" is displayed in the console. This message serves as an indicator of the process.
- **Returns:** The function does not return any value as it is designed to exit, leading to the BIOS booting the local disk.
- **Example usage:** `bash    ipxe_boot_from_disk` ##### Quality and security recommendations

1. A validation procedure should be put in place to confirm the completion of the iPXE boot. This ensures that the BIOS boot is not initiated erroneously, causing a reboot loop.
2. Increase delay if required. The 5-second delay might not be enough in some systems, making it useful to include an optional delay argument.

3. Implement logging mechanism to ensure the function can be debugged and monitored.
4. Consider implementing an error-handling structure to manage potential execution failures. This could help in diagnosing and resolving issues more effectively.
5. Echoing the communications or actions to the end-user without any validation or filtration might pose a security risk. Therefore, sanitization of outputs is recommended.

6.6.0.63 `ipxe_boot_installer`

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

Function signature: bd7740979eac5c8a20bebd6f254b153d1f67f954620f764cd38ca5b87c1ca613

Function Overview The `ipxe_boot_installer` function handles the installation process for a new host based on its type. The function firstly loads server-side profiles for different host types and then retrieves specific parameters needed from these profiles. It also checks if the host is already installed and if it is, the function gets aborted and the host gets a reboot command. Specific host configurations are set and an ISO file containing the distribution of the operating system is mounted. The function supports several operating systems, including Rocky Linux.

Technical Description

- *Name*: ipxe_boot_installer
- *Description*: This function handles the transitioning of a host's state to the installation process preliminaries, such as setting up configurations, checking host statuses, and preparing the actual booting process from server-side profiles and parameters.
- *Globals*: [HPS_DISTROS_DIR: Directory to the distributions of operating systems, CGI_URL: URL to the CGI Script, FUNCNAME: Function Names]
- *Arguments*:
  - $1: `host_type` describes the type of the host that will be installed
  - $2: `profile` specifies the profile to be used for the installation
- *Outputs*: An IPXE_BOOT_INSTALL script for network booting the host server
- *Returns*: Nothing
- *Example usage*:

```
ipxe_boot_installer "rockylinux" "profile1"
```

Quality and Security Recommendations

1. To improve readability of the script, it's recommended to document values and expected outputs of internal function calls.

2. There are few user-oriented error messages. An enhancement would be adding more informative and user-friendly messages.
3. There is a potential security risk as path variables for files and directories are not quoted, thus might lead to command injection vulnerabilities if they contain spaces or special characters. Quoting them could prevent such issues.
4. To increase function modularity, a consideration could be dividing this long script into separate smaller functions.
5. Ensuring regular updates, patches, or security fixes for the operating systems installed on the hosts.

6.6.0.64 `ipxe_cgi_fail`

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

Function signature: d366f9fa44a1d1323d1e8c6c1942275158e673ada8470890bb2f44ffcd619f88

Function Overview The bash function `ipxe_cgi_fail` is dedicated to process iPXe failure messages. This function is part of a broader bash script that deals with errors or unexpected behaviours occurring in an iPXe (network boot firmware) environment. It visualizes, logs and reboots the system in a case of an iPXe related failure. Note that this bash function does not resolve the errors, it merely captures, logs and processes the failure.

Technical Description The `ipxe_cgi_fail` function can be technically dissected as follows:

- Name: `ipxe_cgi_fail`
- Description: This function is involved in processing iPXe failures. It formats and reflects the failure message, logs it, and issues a command to reboot the system.
- Globals: None
- Arguments: `$1` (The failure message to be displayed, logged and processed)
- Outputs:
  - Outputs the failure message and echoes some error-related strings to stdout which eventually get rendered by iPXe.
  - Logs the error message using `hps_log`.
- Returns: None, the function ends with an `exit` command.
- Example Usage:

  ```
  ipxe_cgi_fail "Network failure during boot"
  ```

  This will display and log the mentioned error “Network failure during boot” and then reboot the system.

Quality and Security Recommendations

1. Always make sure to validate and sanitize the input error message before processing it in the function for security reasons.
2. For improved maintainability, consider having separate functions to display, log, and reboot in the event of errors.
3. If there is a risk of sensitive data being output in the error message, handle the error messages in a way that would avoid exposing sensitive data.
4. Consider a more comprehensive error handling feature, beyond merely logging and rebooting the system.
5. If the function is used frequently, consider incorporating an option to configure the sleep time duration or whether the system should reboot or not.
6. Always ensure to test the function thoroughly under different scenarios and potential edge cases to ensure its stability and reliability.

6.6.0.65 `ipxe_configure_main_menu`

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

Function signature: 3dbb119c0559dd930232dc88e23a76f65948543b5080c8b27f769b30f11650ee

Function overview The `ipxe_configure_main_menu` function is responsible for delivering the configure menu when the cluster is configured, and the host is not. This menu is useful in defining a list of options that the user can select from. For instance, host options, system options, advanced options.

Technical description - **Name:** `ipxe_configure_main_menu`

- **Description:** The primary function of `ipxe_configure_main_menu` is to deliver a configuration menu to the client in the scenario where a cluster is configured, but the host is not. It uses an array to specify optional item selection which can be used in further configuration or system processes.

- **Globals:** [`mac`: obtains the Media Access Control (MAC) address of the network device, `FORCE_INSTALL_VALUE`: Determines whether to force installation.]

- **Arguments:** [`$mac`: MAC address specific to the system being configured, `$FUNCNAME[1]`: Returns the name of the calling function]

- **Outputs:** Writes a menu with several configuration options ready for user interaction.

- **Returns:** It outputs the menu status message. If it fails, it returns a log failure message.

- **Example usage:**

```
ipxe_configure_main_menu \$mac $FUNCNAME[1]
```

Quality and security recommendations

1. Always validate the returned values and handle exceptional scenarios through error handling and recovery.
2. Employ debugging mechanisms to identify possible faults or errors.
3. Incorporate a logging mechanism to keep track of all menu interactions. This assists in identifying and resolving any potential issues.
4. Check to ensure that no sensitive information, like the MAC address of the client, is exposed or logged.
5. Commands that interact directly with the system should be reviewed carefully to prevent arbitrary command execution or injection vulnerabilities.

6.6.0.66 `ipxe_header`

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

Function signature: f724cbc120f1a1eae5cf985238d6bc44a932f4521832fa8959c47118c5068ed5

Function overview This function, `ipxe_header`, is used to send a pxe header to prevent boot failure. It sets variables for IPXE scripts and then utilizes these variables within an 'ipxe' script block. The ipxe script block is used to set a log message, fetch an image, and display client and server details.

Technical description

- **name:** ipxe_header
- **description:** The function sends a pxe header to prevent boot failure, sets some variables, fetches an image with the log message, and echoes the client and server details.
- **globals:**
  - `CGI_URL`: The URL to the boot_manager.sh script on the next server.
  - `TITLE_PREFIX`: Prefix for the title containing name of the cluster, MAC address and IP address of the net0 interface.
- **arguments:** None
- **outputs:** Log messages that includes cluster name, client's IP and MAC address
- **returns:** None
- **example usage:** `bash ipxe_header` ##### Quality and security recommendations

1. It is important to properly validate the input for functions to prevent unwanted side effects or attacks. Thus, do ensure that input such as the server and client IP addresses are valid before they are used in server-side scripts.
2. Always use https:// prefix for `CGI_URL` if the data sent within the script is sensitive or important, as http:// is not secure.

3. Beware of command injection security vulnerabilities. All variables that are included in command line arguments should be properly escaped.
4. The function does not implement error checking or handling. It is recommended to handle possible error conditions to improve the robustness of the application. For instance, check if `cgi_header_plain` and `imgfetch` have executed successfully, and if not, throw a meaningful error message.

6.6.0.67 `ipxe_host_install_menu`

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

Function signature: 0627b23e1b7a33e58451ad40a8e31ebc0e9911be4bc534e1afb8dc54dea050c4

Function overview The function `ipxe_host_install_menu` is used in the initial menu of a bootstrapping application. Its key function is to set up a menu for the selection of installation types, specific to each host. After receiving user's choice, it generates a logging message and fetches it from the server, then proceeds to execute the chosen item accordingly. If any error occurs during the logging message fetch attempt, the function will print "Log failed".

Technical description

- Name: `ipxe_host_install_menu`
- Description: Function sets up the installation menu during the bootstrapping phase and allows user to pick the installation type per host. Logs the selection and hands over the control to the selected item.
- Globals: [`FUNCNAME`: tracks the name of the function currently being executed, `TITLE_PREFIX`: holds the prefix string that will be displayed with the title, `CGI_URL`: stores the server URL for fetching the log message, `selection`: holds the selected menu item and `logmsg`: stores the logging message]
- Arguments: None
- Outputs: Logs the user's selected choice for installation and executes the relevant installation process.
- Returns: No explicit return. Implicitly returns the status of the last executed command.
- Example usage:

```
ipxe_host_install_menu
```

Quality and security recommendations

1. Always set up a fail-safe option to catch any unwanted choices or accidental key presses.
2. To ensure security, use an encrypted or secure connection when fetching messages from servers.

3. When using global variables, make sure these do not clash with other existing global variable names.
4. Check the necessity of all globals. If the usage of the global variable can be replaced by local variables or function parameters, opt for those.
5. Provide user feedback whenever critical operations happen like fetching data from server successfully or unsuccessfully.
6. Do not expose critical configuration info or sensitive data in any log or error message.
7. If possible, consider implementing some sort of menu item validation to ensure the selected option is a valid one.

6.6.0.68 `ipxe_host_install_sch`

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

Function signature: 76d8d63df923bfeaa3d9b18625e12e7909180d679baee564a7d5760de6c84baa

Function Overview The `ipxe_host_install_sch` function is designed to manage a user interactive menu necessary for installing the Storage Cluster Host (SCH). It fetches the log message and menu selected by the user, executes the installation immediately, and manages various other actions such as moving back to the initialisation menu, and different installation options like ZFS single-disk or ZFS RAID Installation.

Technical Description

- **Name**: `ipxe_host_install_sch`
- **Description**: This function manages a menu which facilitates the installation of the Storage Cluster Host (SCH). It offers various options such as backing to the initialization menu and selecting the type of installation. It fetches and logs the menu selected by the user and executes the selected item.
- **Globals**:
  - `TITLE_PREFIX`: Describes the prefix for all titles in the menu.
  - `CGI_URL`: The URL to fetch and process commands.
- **Arguments**: None
- **Outputs**:
  - Presents an interactive menu to the user.
  - Sends a fetch request to the log with `imgfetch`
  - Chains processes with direct command requests using `chain --replace`
- **Returns**: None
- **Example usage**: `ipxe_host_install_sch`

Quality and Security Recommendations

1. Validate all external inputs to the function to prevent code injection or data corruption as the function directly interacts with an external URL.
2. Confirm that the URL (`CGI_URL`) is secured using HTTPS to ensure all communications are encrypted.
3. Incorporate error handling for all network requests in the function to ensure the program is robust and less prone to crashing.
4. Regularly update and patch the function to patch any vulnerabilities and avoid using deprecated functions.

6.6.0.69 `ipxe_init`

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

Function signature: b99f8e1adfe13d429e91c39da6aeab71b263f99c8d73aa5f35dd7351988760e2

Function Overview The function `ipxe_init()` initializes iPXE for network booting. This function is triggered only when the cluster is already configured and we do not yet know the exact MAC address of the host. The function effectively works by sending requests to a URL, attempting to fetch and load a config file, and then executing this config. If no config file can be found, the script displays an error message and reboots. However, this scenario should not occur as the boot manager creates the configuration file.

Technical Description

- **Name**: `ipxe_init`
- **Description**: This function is used for initializing iPXE for network booting by trying to fetch and load a server-side provided configuration. This function is only applied when the environment has been set up and the MAC address of the host has not been determined.
- **Globals**: [`CGI_URL`: The URL which generates the network booting configuration based on a specified MAC address and boot action command]
- **Arguments**: none
- **Outputs**: The function will output a request to `config_url` and the configuration status after attempting to load it. If no host config is found, it will output an error message and execute a system reboot.
- **Returns**: No specific return value
- **Example usage**:

```
ipxe_init
```

Quality and Security Recommendations

1. Comment unexplained code: There are several parts of the function (#|| goto no_host_config, #:no_host_config) that are commented out. If this code is not

   required, it should be removed. Otherwise, the functionality of this code should be clearly explained in comments.
2. Implement error handling: The function should be able to handle error situations more gracefully. For example, the function could try alternative ways to fetch and load the config file if the initial attempt fails.
3. Security improvements: Ensure that the URL fetched from the `CGI_URL` variable is a trusted source to prevent potential security risks.
4. Implement logging: The function could incorporate logging to record any issues which occur during the execution. This would assist with effective debugging and problem resolution.
5. Follow naming conventions: All variable names should adhere to a consistent naming convention. This increases readability and maintainability.

6.6.0.70 `ipxe_reboot`

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

Function signature: 29b797dcc656b196df87e13db53986bcc51b8bae19cae708c8d71b4ed213e555

Function Overview The function `ipxe_reboot` is a bash function that is used to log a message about a requested reboot and then perform the reboot after a 5 second delay. It accepts a single argument, which represents the message to be printed before the reboot occurs. It also uses the `ipxe_header` function, although it's unclear from the provided snippet what this function does.

Technical Description

- Name: ipxe_reboot
- Description: This function logs a message, prints an additional message, waits 5 seconds, and then triggers a system reboot.
- Globals: `mac`: A global variable that contains the Mac address of the machine. Used in the log message.
- Arguments: `$1: MSG`: A message to be printed before the reboot takes place.
- Outputs: The function outputs several echo commands, including the passed message and the string "Rebooting…"
- Returns: Not applicable, as the function doesn't return a value but causes the system to reboot.
- Example usage:

```
ipxe_reboot "System update completed"
```

In this case, the string "System update completed" is logged and printed before a system reboot is triggered.

Quality and Security Recommendations

1. The `ipxe_header` function is invoked without any context or explanation, which may confuse other developers trying to understand the code. It is recommended to clarify its role within this function.
2. This function directly includes the value of `$MSG` within a string that it `echo`s. This has the potential to cause security vulnerabilities, known as Shell Injection, when uncontrolled input is passed as a message. It would be advisable to sanitize this input before rendering it in this way.
3. Similarly, it appears that the `mac` global variable is used without any validation or sanitization. As this appears to be a Mac address, it would be wise to validate that it fits the expected pattern for such an identifier.
4. The hardcoded `5` second delay before reboot could be made more flexible. It might be beneficial to allow this delay to be an optional argument to the function, allowing the caller to specify a different time period if needed.
5. There are no checks to ensure that the current user has appropriate permissions to execute a reboot. Checking these permissions before attempting a reboot would improve the stability and error handling capabilities of this function.

6.6.0.71 `ipxe_show_info`

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

Function signature: b455b51b9b86aa994035c826d4a4341e071bdd8322ff22c818e2e91f4e188387

1. Function Overview The function `ipxe_show_info()` is a bash function that shows host, iPXE system, cluster configuration and system paths information based on the category provided. Upon calling the function, it takes in a category (i.e., `show_ipxe`, `show_cluster`, `show_host`, or `show_paths`) and displays the corresponding information. If a path or necessary file is missing it will indicate this to the user.

2. Technical description

- **Name:** `ipxe_show_info()`
- **Description:** This function displays information about iPXE system, host, cluster configuration and system paths based on the category provided.
- **Globals:** [`HPS_CLUSTER_CONFIG_DIR`: This is the directory where the cluster configuration files are stored. `HPS_CONFIG`: This is the configuration file for HPS.]
- **Arguments:** [`$1`: The category of information to be displayed. It can be `show_ipxe`, `show_cluster`, `show_host`, or `show_paths`]
- **Outputs:** Prints the iPXE system information, host information, cluster configuration or system paths based on the argument provided to the function.
- **Returns:** Nothing.
- **Example usage:** `ipxe_show_info show_cluster`

3. Quality and security recommendations

1. **Input validation:** Ensure that user inputs are properly validated to avoid unexpected behavior or output from the function. For instance, validating the `category` should only take the allowed arguments (`show_ipxe`, `show_cluster`, `show_host` or `show_paths`).
2. **Error Handling:** Provide clear and specific error messages for the end user. In case a configuration file is not found, provide guidance on next steps or tips on how to solve the issue.
3. **Documentation:** Keep the documentation of this function up to date, as it forms a critical part of the user guide.
4. **Security:** Be wary of command execution vulnerabilities if user input is ingested without validation or sanitization. In this function, make sure that the `category` input does not open up the potential for Command Injection. This is particularly important if additional components will be added to the system that calls this function and could open up new vulnerabilities.
5. **Code Quality:** While not directly related to security, maintaining good code quality is a standard best practice to keep a project maintainable and error free in the long term. Code quality refers to such things as ensuring consistent syntax style, comprehensive commenting, avoiding deeply nested loops or conditional (i.e., “spaghetti code”), and dividing code into modular, single-purpose units.

6.6.0.72 `list_clusters`

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

Function signature: 93e2d81ba91f8f27fd695171a6ba9261942077d5e68bbf9699a91e819f6fdd9e

1. Function Overview The `list_clusters` function is designed to list and display all clusters located within the configured base directory of a High Performance Server’s (HPS) clusters. This function makes use of the `nullglob` Bash option, which allows for the prevention of errors when no files or directories match the provided pattern. The function iterates through each cluster in the base directory and echoes its base name to the standard output.

2. Technical Description

- **Name:** list_clusters
- **Description:** This function is used to list all clusters present in the base directory set by `HPS_CLUSTER_CONFIG_BASE_DIR`.
- **Globals:** `HPS_CLUSTER_CONFIG_BASE_DIR`: This variable describes the base directory where the HPS clusters are located.
- **Arguments:** No arguments are required to be passed to this function.

- **Outputs:** The base names of the clusters located in the `HPS_CLUSTER_CONFIG_BASE_DIR` directory.
- **Returns:** It does not explicitly return a value, but echoes names of clusters on standard output.
- **Example Usage:**

```
list_clusters
```

3. Quality and Security Recommendations Given the function simply outputs the names of directories, it poses minimal security threats. However, a few general recommendations can be made to ensure best practices:

1. **Access Control:** The script should ensure that proper access controls are placed on the base directory. The script should also verify the reader's permission before access.
2. **Path Sanitization:** If there is user interaction before this function, ensure that path values coming from users are sanitized to prevent potential directory traversal attacks.
3. **Error Handling:** The script should handle scenarios where the base directory doesn't exist or couldn't be read, as well as when there are no subdirectories present.
4. **Documentation:** While this function is straightforward, documenting its expectations and output can help with usage and troubleshooting.
5. **User Feedback:** Providing a feedback when no clusters are found might enhance the user experience.

6.6.0.73 `list_local_iso`

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

Function signature: 2551c5d34fea622a4876a48c635391772020f951a812cc21a423cc6b87227a67

Function overview The `list_local_iso()` function is designed to search and list iso files that match a specific naming pattern in a certain directory. The naming pattern derives from the CPU, manufacturer, operating system name and optionally, the operating system version. Important directories and files are managed with safety checks. Results of iso file search are displayed to the user with an alert mechanism implemented in case no matching iso files are found.

Technical description **Name:** `list_local_iso()`

Description: This function searches for iso files in a specific directory that match a naming pattern based on the CPU, manufacturer, operating system name and optionally, the operating system version.

Globals: - `iso_dir`: the directory on the local system where the iso files are stored

Arguments: - `$1: cpu`: The CPU model - `$2: mfr`: Manufacturer information - `$3: osname`: Name of the operating system - `$4: osver`: (Optional) Operating system version

Outputs: Prints out the name of any iso files found that match the naming convention.

Returns: 1 if no iso files were found that match the naming convention.

Example usage:

```
list_local_iso "Intel" "Dell" "Ubuntu" "18.04"
```

Quality and security recommendations

1. Add error checks for input parameters like invalid CPU, manufacturer, or OS name to enhance usability.
2. Implement permission checks before accessing `iso_dir` directory to prevent unwarranted access.
3. Enhance the pattern matching logic to prevent unintended matches.
4. Include logging for tracking and troubleshooting.
5. Improve the return codes mechanism to provide more specific error messages in case of failure.
6. Introduce a silent mode for suppressing output whenever needed. This will be helpful during automated processes.

6.6.0.74 `load_cluster_host_type_profiles`

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

Function signature: dc36e1e312e63f09fcc2eec91fb9e4bba6f60eb0ff592f0a1648a9a56bd34a80

1. Function Overview This bash function, `load_cluster_host_type_profiles()`, is responsible for loading cluster host type profiles from a configuration file. The function reads from the active cluster configuration file (which name is retrieved by `get_active_cluster_filename()` function) and declares the host types with their corresponding key-value pairs. These key-value pairs are then stored in a global associative array. If the configuration file does not exist, the function returns 1 without any execution.

2. Technical Description **Function Name:** `load_cluster_host_type_profiles`

Description: This function reads from an active cluster's configuration file and creates a set of declared host types with their corresponding key-value pairs. This information is then stored in a global associative array for later use.

Global Variables:

- `__declared_types`: holds an associative array of declared types of hosts.

Arguments: None.

Outputs: If successful, the global associative array, `__declared_types` is populated with host types and their corresponding key-value pairs from the active cluster configuration file.

Returns: If the configuration file does not exist, the function returns `1`. Otherwise, it doesn't return a specific value.

Example Usage:

```
load_cluster_host_type_profiles
```

3. Quality And Security Recommendations

1. All crucial actions like creating files or reading from them should have error handling to ensure no unexpected behavior occurs.
2. Variables such as `config_file` should be properly sanitized to prevent potential code injection attacks.
3. A detailed and meaningful error message should be provided if the active cluster configuration file is absent.
4. Implement file access permission check for the configuration file to avoid unauthorized changes.
5. If the function is going to be used in multi-threaded scenarios, proper measures should be taken to avoid data races and inconsistencies concerning global variable `__declared_types`.

6.6.0.75 `make_timestamp`

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

Function signature: 7ea1fc9d3621ad0a04879323d75cd0a5f1aa2468bf98b9b3c83ff2b66dfa8e3d

Function overview The presented function, `make_timestamp()`, is used to generate a string representing the current date and time in Coordinated Universal Time (UTC). The string follows the format: "YYYY-MM-DD hh:mm:ss UTC".

Technical description

- **Name:** make_timestamp
- **Description:** A bash function that generates a timestamp in the format 'YYYY-MM-DD hh:mm:ss UTC' using the current date and time in Coordinated Universal Time (UTC).
- **Globals:** None
- **Arguments:** None

- **Outputs:** Prints the generated timestamp to standard output.
- **Returns:** `0` if the command was successful. Otherwise, it depends on the error code of the `date` command.
- **Example usage:**

```
$ make_timestamp
2021-09-30 10:25:30 UTC
```

Quality and security recommendations

1. Avoid storing timestamps in plain text to prevent data leaks.
2. This function relies on the `date` command's success. There is no error handling if `date` fails. Adding error checking can make the function more robust.
3. To ensure the correctness of the timestamp, it is recommended to synchronize the system clock with a trusted time source.
4. This function doesn't take any arguments. Although this protects against argument-related security vulnerabilities, the function could be expanded to accept input if flexibility is needed.

6.6.0.76 `mount_distro_iso`

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

Function signature: 9ee707a09f131340e40ead1764695e3c9db4201a8c595ad06c9a5f7028376217

Function overview The function `mount_distro_iso()` is used in Bash scripts to mount a digital distribution image (also known as an ISO file) of a particular distribution, specified by the distribution string parameter. This function checks if the ISO file exists and if the mount point is already in use. If the conditions are satisfied, the function mounts the ISO file to the mount point.

Technical description **Name:** `mount_distro_iso`
Description: Mounts a specific distribution's ISO file to a dedicated mount point.
Globals: `[ HPS_DISTROS_DIR: The directory containing distribution ISOs and their respective mount points. ]`
Arguments: `[ $1: The distribution string of the target ISO, $2: (optional, not used) ]`
Outputs: Log messages indicating the process.
Returns: It returns `1` if ISO is not found, `0` if mount point is already mounted, nothing if successful.
Example usage: `mount_distro_iso "Ubuntu"`

Quality and security recommendations

1. Parameter Validation: Consider enhancing the function by adding more validation to the parameters. Right now, missing or incorrect arguments could lead to unwanted behavior.
2. Error Handling: There's no error handling if the `mount` command fails. Adding an error handler for this scenario could enhance the quality of the script and provide more information for debugging.
3. Security Review: Review the function to ensure the script handling sensitive input correctly. Keep in mind that improperly handled inputs can result in security issues.
4. Path Validation: Ensure the paths used in the script are safe and can't lead to a path traversal attack.
5. Logging: Enhance logging to include more useful information for debugging, like wrong argument values or file path-related issues.

6.6.0.77 `normalise_mac`

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

Function signature: ac8cefca0a4fe56f9e4ef01e54a13bb17bd1107670d5f1b98c4c04d07fd2425e

Function Overview The function `normalise_mac()` is utilized to sanitize and validate a given MAC address input to be certain it adheres to the standard format of a 12-character hexadecimal string. It initially removes all common delimiters, such as colons, hyphens, periods, and spaces, converts the string to lowercase, and validates that the sanitised input is exactly a 12-digit hexadecimal string. If the input fails to meet these criteria, the function echoes an error message and returns 1 (indicating failure of the function). If the input is validated, the function echoes the sanitized and validated MAC address.

Technical Description

- **name**: normalise_mac
- **description**: This bash function sanitizes and validates a given MAC address to ensure it adheres to the 12-character hexadecimal string format.
- **globals**: None
- **arguments**:
  - $1: Input String - This is the MAC address being sanitized and validated.
- **outputs**: The sanitized and validated MAC address if input is valid, otherwise an error message indicating an invalid MAC address format.
- **returns**:
  1. If the MAC address input is invalid
  2. If the MAC address input is valid
- **example usage**: `normalise_mac "78-30-15-AB-90-67"`

Quality and Security Recommendations

1. Code Comments: For better readability and maintainability, each block of code could be introduced with a descriptive comment explaining its purpose.
2. Input Validation: The input is sufficiently validated, but more comprehensive validation could be implemented to strengthen error handling, for instance checking for null or empty strings.
3. Test Cases: Incorporate test cases to verify that the function correctly sanitizes, validates and handles various types of MAC addresses inputs.
4. Error Handling: Considering more advanced error handling mechanisms such as a try/catch block could additionally be beneficial for troubleshooting and debugging.
5. Exit Codes: Using standardized exit codes could help make the function more universal and better interact with other scripts or applications.

6.6.0.78 `prepare_custom_repo_for_distro`

Contained in `lib/functions.d/repo-functions.sh` Function signature: f41fadc94c0ebb53ebb87776324ccfcc73903f665dfe9bfd0a9d3f4c9cbb827c

Function overview The function `prepare_custom_repo_for_distro` is designed to prepare a custom repository for a specified Linux distribution. The function takes in a distribution string and any number of package sources or names. It identifies file sources and package names, creates a repository directory, and then either downloads or copies packages as necessary to the created directory. Finally, it builds the repository and verifies the necessary packages are included.

Technical description

- **Name**: `prepare_custom_repo_for_distro`
- **Description**: This Bash function prepares a custom repository for a specific Linux distribution. It identifies package sources and names, creates the repository directory, downloads or copies package files to the directory, builds the repository, and verifies the required packages.
- **Globals**: There's one global variable used, namely `HPS_PACKAGES_DIR`. This is the base directory where the custom distribution-specific repository directory will be created.
- **Arguments**:
  - `$1`: The distribution string that will be used to create a distribution-specific directory within the base repository directory.
  - `$@`: An array of package sources and/or names.

- **Outputs**: Text information about the ongoing process is logged using `hps_log` function and error messages are logged if something goes wrong creating the directory, downloading or copying packages, building the repository, or verifying the required packages.
- **Returns**: The function has different return codes based on the kind of error it encounters. They are as follows:
  - 1 if there is a failure in creating the repository directory.
  - 2 if there's a failure in downloading a package.
  - 3 if there's a failure in copying a local file.
  - 4 if there's an invalid package source.
  - 5 if the building of yum repo metadata fails.
  - 6 if there is a missing required package in the repository. If the function executes successfully, it returns 0.
- **Example usage**: `prepare_custom_repo_for_distro ubuntu16 hhttps://package1.com package2`

Quality and security recommendations

1. Input Validation: You should validate the inputs to the function to ensure they are not null, and that they meet any other necessary criteria (e.g., distribution string is from a known set).
2. Error Handling: Add error handling for more return conditions to ensure the function doesn't execute unnecessary steps or gives a clear and exact error message when it fails.
3. System Commands: Carefully review your use of system commands such as `mkdir` and `cp`. Consider adding checks to confirm whether files and directories exist before you try to create or copy them. Also, consider the possible security implications of using these commands, as injecting malicious input could lead to unintended behavior.
4. Logging: Add more logging at each critical points of the function execution to better understand the behaviour. This can be especially useful in understanding unexpected failures in the future.
5. Global Variables: The usage of global variables could lead to unexpected behavior if they are modified elsewhere in the script. Consider passing `HPS_PACKAGES_DIR` as an argument to the function if possible.

6.6.0.79 `print_cluster_variables`

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

Function signature: c5583d8fd4715e19ec442b98a50c60b43c9b20d0f5892f8d147bbb29263d00fa

Function Overview The `print_cluster_variables()` function in Bash reads a file containing configuration parameters for a cluster and prints these variables to the standard output. It takes the configuration filename as its only argument. As it processes the file, it skips blank lines and lines beginning with a hash mark (#), which are interpreted as comments. If the file isn't found, the function prints an error message to the standard error and returns 1. Variables are unquoted before they are printed.

Technical Description

- **Name**: `print_cluster_variables()`
- **Description**: This function reads a cluster config file, unquotes variable values and prints them to stdout. If the config file is not found, an error message is printed to stderr.
- **Globals**: Not applicable.
- **Arguments**:
  - `$config_file`: Refers to the name of the configuration file for the cluster, obtained from the function 'get_active_cluster_filename'.
- **Outputs**: This function will output each key-value pair from the configuration file, or an error message if the file does not exist.
- **Returns**: `1` in case of error (if the config file does not exist); no specified return in case of successful operation.
- **Example Usage**:

  ```
  print_cluster_variables
  ```

Quality and Security Recommendations

1. Always check that provided input is valid. In the current implementation, a file is considered to be a valid configuration file if it exists, but no check is performed to ensure that it contains valid content.
2. Implement error handling for file operations. Currently, if the config file can't be read for any reason other than nonexistence (such as insufficient permissions), the function might behave unpredictably.
3. Only important outputs are being sent to stdout, whilst messages are being sent to stderr. This differentiates between types of outputs making it easier to redirect and handle all types of outputs effectively.
4. Document the function thoroughly: Though the function has a simple and clear name, it would be beneficial to add comments in the code to clarify what it does, how it should be used and what kind of inputs/outputs it expects/deals with.

6.6.0.80 `register_source_file`

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

Function signature: 3a47ee217d4c21d4f884f60289c0b97418f2bea9352f1117fe8517c7bb30bb0b

Function overview The Bash function `register_source_file()` takes two arguments: a `filename` and a `handler`. It registers a source file in the `dest_dir` directory by appending the file and handler name to the `index_file`. If the entry has already been registered, an informational message gets printed, and the function returns without making changes. If the destination directory doesn't exist, it gets created automatically.

Technical description

- **Name:** `register_source_file`
- **Description:** The function registers a source file in a specific directory designated as `dest_dir`. It avoids duplicate entries and provides feedback messages during the process.
- **Globals:** `[HPS_PACKAGES_DIR: The directory where the packages will be stored. If this variable is unset, the directory '/srv/hps-resources/packages' is used.]`
- **Arguments:** `[$1: The filename to be registered, $2: The respective handler for the incoming file]`
- **Outputs:** Prints messages to standard output indicating whether the file already existed or was successfully registered.
- **Returns:** Returns 0 if the source file is already registered. There is no other explicit return value, so if execution reaches the end , Bash will return the exit status of the last command, which is expected to be 0.
- **Example usage:**

```
register_source_file "myFile.txt" "myHandler"
```

Quality and security recommendations

1. Input validation: Add checks to ensure the filename and handler are not empty before proceeding with the function.
2. Error handling: Monitor the `mkdir -p "$dest_dir"` and `echo "${filename} ${handler}" >> "$index_file"` statements for possible failures. Exit the function and report an error if these statements fail.
3. Path traversal check: To improve security, check that the filename argument does not contain any directory traversal components like "../" or unexpected special characters.
4. Avoidance of global variables: Instead of using the global variable `HPS_PACKAGES_DIR`, consider adding it as third parameter to the function.
5. Permissions check: Ensure that the script runs with the necessary filesystem permissions to create directories and modify files in the destination directory.

6.6.0.81 `reload_supervisor_config`

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

Function signature: 1ad96e9487bac5b94d7eca312662f18392454507c7cddfac5de26d924e922722

Function Overview The function `reload_supervisor_config` is designed to re-read and update the Supervisor daemon configuration. This is useful for ensuring that the configuration changes in the Supervisor are recognized by the system and take effect without needing to restart the entire daemon.

Technical Description

- **Name:** `reload_supervisor_config`
- **Description:** This function updates the supervisord configuration by reading and updating it using `supervisorctl` with the configuration file specified by the `HPS_SERVICE_CONFIG_DIR` environment variable.
- **Globals:** [`HPS_SERVICE_CONFIG_DIR`: Environment variable specifying the directory where the `supervisord.conf` configuration file is located]
- **Arguments:** None
- **Outputs:** Outputs from the reread and update commands, typically messages about any changes in the configuration.
- **Returns:** Outputs the status of configuration re-read and update, it does not explicitly return a value.
- **Example usage:**

  ```
  reload_supervisor_config
  ```

Quality and Security Recommendations

1. Always ensure that `HPS_SERVICE_CONFIG_DIR` is a valid directory and has the correct permissions to avoid an access issue.
2. Add error handling for scenarios where the `supervisord.conf` file may not exist or `supervisorctl` commands might fail.
3. Be mindful of managing your Supervisor configurations and regularly check and update them as needed.
4. Ensure the process running this script has rights to restart or reload supervisor services. This function might be dangerous if it is run by a process with elevated permissions.

6.6.0.82 `remote_function_lib`

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

Function signature: b6d9cd335e4f61b186f614aa07279b2f8bdd77298bf573a231ffd21869c18118

Function overview This function `remote_function_lib` is used to inject pre and post sections into a specific file. It utilizes the concept of here documents in bash scripting to create static blocks of text. The function reads from the "EOF" (End Of File) marker until it encounters another "EOF" marker, taking the input between both markers and passing it as an output. This is often used for multi-line strings or scripting inside scripting.

Technical description

- **name:** remote_function_lib
- **description:** A function that uses the here document syntax («EOF … EOF) to inject pre and post segments into a specific file. It could be very useful when you need to append static multi-line text into files or scripts.
- **globals:** None
- **arguments:** None
- **outputs:** Injects specified text into a given file
- **returns:** None
- **example usage:**

  ```
  remote_function_lib > target.sh
  ```

  This example will redirect the contents generated by `remote_function_lib` to a file named `target.sh`.

Quality and security recommendations

1. The here document syntax could potentially lead to command injection if incorrect data is supplied. Be sure to validate input data.
2. The cat command can result in displaying or injecting of unintended data. Streamlined error handling could prevent misdirected data from being displayed.
3. It is recommended that repetitive tasks such as this function are placed in their own script file. This will promote code reusability and maintainability.
4. It is unclear as to what exactly the pre and post sections would contain. Having clear documentation about the expected contents can improve understandability and usability.
5. The function could return a status or a value indicating its success or failure which would improve error handling capabilities.

6.6.0.83 `remote_log`

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

Function signature: 1f586056ba1b573eed69d32639c3940c1c742ba73af4aae917a1d65d36c5367c

Function Overview The function `remote_log()` is designed to URL encode a message from the input parameter and send it as a log message to the specified gateway. The log message is sent via a HTTP POST request to a Bash script called `boot_manager.sh` on the gateway host.

Technical Description

- Name: `remote_log()`
- Description: This function takes a message as input, URL encodes the message, and sends a HTTP POST request to send the message to a gateway host.
- Globals: [`macid`: Identifier for the machine, `HOST_GATEWAY`: Address of the gateway host]
- Arguments: [`$1`: the message to be logged and sent, `$2`: Not used]
- Outputs: None, this function does not produce any output.
- Returns: It doesn't return any value.
- Example usage:

```
message="This is a test message"
remote_log "$message"
```

Quality and Security Recommendations

1. User input should be sanitized before being passed into the function to avoid potential security vulnerabilities.
2. Unused parameters (such as `$2`) should be removed to avoid confusion and maintain cleaner code.
3. Any potential errors from the `curl` command should be handled gracefully. Consider adding error checking to ensure that the message has been sent successfully.
4. Always ensure that the `macid` and `HOST_GATEWAY` variables are properly set and valid.
5. Consider using `https` instead of `http` for the POST request to enhance the security of the function.

6.6.0.84 `rocky_latest_version`

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

Function signature: f42e139a07133ae36c4e9783c1fbb144e1167e59537bd374c0b346f853c77a3d

Function Overview The `rocky_latest_version` function is used to retrieve the current latest version of Rocky Linux. The function does this by issuing a cURL request to the official Rocky Linux download page, parsing the returned HTML for version numbers and sorting them. The function then returns the highest version number.

Technical Description:

- **Name**: rocky_latest_version
- **Description**: This function retrieves the latest version number of Rocky Linux by parsing the HTML from the official Rocky Linux download webpage.
- **Globals**:
  - base_url: The URL to the official Rocky Linux download webpage
  - html: Stores the HTML content retrieved from the Rocky Linux download page
  - versions: An array that stores the version numbers extracted from the HTML content
- **Arguments**: This function does not take any arguments.
- **Outputs**: If successful, the function will output the latest version number of Rocky Linux.
- **Returns**:
  - 1: If the cURL request fails or no version numbers are found in the HTML content
  - The latest version of Rocky Linux: If the cURL request is successful and version numbers are found in the HTML content
- **Example Usage**: `bash      rocky_latest_version`

Quality and Security Recommendations

1. Always use the `-r` (raw) option with the `readarray` or `mapfile` Bash built-ins to avoid problems with backslashes. A `-t` option could also be added to remove trailing newlines.

2. The function could be improved by adding error handling for the case where the Rocky Linux download page URL changes or becomes inaccessible.

3. For critical applications, avoid parsing HTML with regex carefully. Instead, use a proper HTML parsing tool or API if available.

4. Consider validating the output of this command before using it in your script. For instance, you could check that the format of the returned version number matches your expectations.

5. Avoid logging sensitive information. Since this function doesn't handle sensitive data, it's not a concern in this case, but it's a good general practice when writing Bash scripts.

6.6.0.85 `script_render_template`

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

Function signature: 064ddcb49f3688c1b0ede8ce884eae36c9ac96f5117338bdce1f62d5c6960a67

Function overview The `script_render_template` function remaps all placeholders in the form of `@...@` with the corresponding values derived from `${...}`. This function uses environment variables and `awk-vars` for its placeholder values: if a placeholder does not exist in `awk-vars`, the value will default to an empty string.

Technical description

`script_render_template`

- **Description**: This function remaps all `@...@` placeholders with their corresponding `${...}` values. If a placeholder does not exist in `awk-vars`, it uses an empty string as the default value.
- **Globals**: None.
- **Arguments**: None. The function processes all global variables available at runtime and uses them to replace placeholders in the script.
- **Outputs**: This function will print a string where every placeholder `@...@` has been replaced with the corresponding `${...}` value.
- **Returns**: This function does not explicitly return a value. However, it prints a line (or lines) with replaced placeholders, which can be used as a return in Bash.
- **Example usage**:

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

Quality and security recommendations

1. Use descriptive variable names that accurately reflect the data they hold.
2. Validate all data before using this function to ensure safety and accuracy.
3. Avoid creating global variables unnecessarily, especially if they are only used in this function.
4. Use secure methods for sourcing the variables used in this function, such as exporting only required variables.
5. To avoid unintended script behavior, ensure all placeholders are correctly formatted as `@...@`.
6. Document the usage of this function, as it has implicit dependencies on environment variables.
7. Regularly audit and update the function to maintain its security and reliability.

6.6.0.86 `select_cluster`

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

Function signature: 61156c9ba1a9860121b17799a317ed88f6eb231e11490aec4753746b8bc69cb9

Function Overview The `select_cluster` function searches for clusters in a specified base directory, and prompts the user to select one of the available clusters. If no clusters are found in the base directory, the function prints a message to standard error and returns with an error status.

Technical Description

- **Name:** `select_cluster`
- **Description:** The function seeks for clusters within a specified base directory. If clusters are found, it will prompt the user to select one. If no clusters are found, an error message is displayed and an error status is returned.
- **Globals:**
  - `HPS_CLUSTER_CONFIG_BASE_DIR`: The base directory where the function looks for clusters.
- **Arguments:**
  - The function does not take any arguments.
- **Outputs:**
  - In case of no clusters being found, the function outputs: `[!] No clusters found in $base_dir` to stderr.
  - Asks the user to select a cluster: `[?] Select a cluster:`.
- **Returns:**
  - If no clusters are found, it returns `1`.
  - If a cluster is selected, it echoes the selected cluster's path and returns.
- **Example usage:**

```
select_cluster
```

Quality and Security Recommendations

1. **Error handling:** The function could benefit from more rigorous error handling. For instance, it may be desirable to check whether the base directory exists and is readable before attempting to look for clusters.
2. **Input validation:** Although this function does not take any arguments, if future modifications lead to user-provided inputs, proper validation should be incorporated to prevent command injection attacks.
3. **Logging:** Consider adding logging statements to track the execution flow and any potential errors of the function for easier debugging and auditing.
4. **Documentation:** All changes, no matter how minor, should be documented to keep the function's description up-to-date and help future developers understand its operation.

6.6.0.87 `set_active_cluster`

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

Function signature: c4f4f433c7e18e128366307bf60133a4e18e81d677c586a39bfe81ef13c6d637

Function overview `set_active_cluster()` is a Bash function that sets a target cluster as an active one in the system. It checks if the provided cluster name corresponds to an existing cluster directory and if there is a configuration file within the said directory. In case of successful validation, it creates a symbolic link to denote the active cluster.

Technical description

- **name**: `set_active_cluster`
- **description**: This function sets a specific cluster as "active" by creating a symbolic link in the base directory. It performs validations to ensure the existence of the cluster directory and configuration file.
- **globals**: [`HPS_CLUSTER_CONFIG_BASE_DIR`: The base directory where all clusters and their configurations reside]
- **arguments**: [`$1: cluster_name` - Specifies the name of the target cluster to be set as active]
- **outputs**: Prints error messages to stderr if validation fails. In case of success, prints an acknowledgement message to stdout with the name of the set active cluster.
- **returns**: Returns `1` if the cluster directory does not exist, `2` if the configuration file `cluster.conf` is not found, and implicitly `0` by default if the operation is successful.
- **example usage**: `set_active_cluster "my_cluster"`

Quality and security recommendations

1. Provide better handling for failure conditions. This includes more verbose output for error messages and setting return codes for different types of failures.
2. To ensure the usage of this function is safe, the validation could be extended to ensuring the user has appropriate permissions to create the symbolic link.
3. Consider validating the contents of the `cluster.conf` file to ensure integrity and applicability of settings.
4. Care should be taken to properly quote variables to prevent issues with whitespace or special characters in filenames.
5. Avoid global variables if not needed (like `HPS_CLUSTER_CONFIG_BASE_DIR`), use function arguments to provide required information.

6.6.0.88 `ui_clear_screen`

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

Function signature: 0493ca3490c6e95475fb08d2f387298f827857616ec67718e0dd7bc3268a31d2

Function overview The `ui_clear_screen()` function is designed to clear everything currently visible on the terminal screen. It does this by calling the `clear` command, or if that command isn't available, it uses a printf invocation to send a special sequence (`\033c`) to the terminal that instructs it to clear the screen. This function is typically used to prepare the terminal for presenting a new set of output information.

Technical description

- **name**: `ui_clear_screen()`
- **description**: A function that clears all visible content on the terminal screen.
- **globals**: None used in this function.
- **arguments**: This function does not accept any arguments.
- **outputs**: Employs a built-in `clear` command or `printf` to output a special sequence, which leads to the terminal screen becoming blank.
- **returns**: Does not explicitly return any value, but its execution results in either a cleared screen or nothing if both `clear` and `printf` fail.
- **example usage**:

  ```
  ui_clear_screen
  ```

Quality and security recommendations

1. Start by adding some error handling for the case where both `clear` and `printf` fail.
2. Clearing the screen may remove valuable information that the user might still need. Consider allowing the user to choose whether they want the screen cleared or not.
3. Always use full paths to binaries (e.g., /usr/bin/printf) to reduce the risk of executing the wrong binary in case a malicious user tampered with system path variable.
4. Include inline comments in complex parts of the function to ensure maintainability.
5. Make sure to test the function in various terminal environments and properly document any differences in behavior.

6.6.0.89 `ui_menu_select`

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

Function signature: 228d39d076d4308652684c61dd46d71781022466ed70155a37a899acdc69da71

Function Overview The function `ui_menu_select()` is used to generate an interactive selection menu in a command-line interface. The function accepts an initial prompt as a primary argument and an array of options. The menu generated by the function allows a user to select an option by entering the corresponding number. If a valid selection is made, the function outputs the selected option and it returns success. If an invalid selection is made, an appropriate error message is generated.

Technical Description **Function Definition: ui_menu_select**

- **Name:** ui_menu_select
- **Description:** Created to generate an interactive selection menu within a CLI (Command-Line Interface).
- **Globals:** None.
- **Arguments:**
  - `$1`: This is a string that is used as the initial prompt in the menu.
  - `$@`: Contains all arguments of the script or function disregarding the first argument that is a prompt.
- **Outputs:** Prompt for the menu options, either the chosen option (if valid), or an error message (if invalid).
- **Returns:**
  - Successful (0) when a valid option is chosen.
  - The function will not return if an invalid option is chosen.
- **Example Usage:** `options=("Option 1" "Option 2" "Option 3") ui_menu_select "Please choose an option:" "${options[@]}"`

Quality and Security Recommendations

1. It's advisable to include error handling for cases when no options are provided.
2. The function currently allows for any integer input as a valid selection. It would be safer to only accept inputs that correspond to presented options.
3. To improve the robustness, consider limiting the total number of invalid attempts before automatically terminating the function to prevent potential misuse.
4. The function echoes the input which is a potential security risk (Command Injection). Consider revising this for improved security.

6.6.0.90 `ui_pause`

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

Function signature: 1636563cd29eae7fb011743bbb84e1bd4b67567168166cfc3cd0cf46472383ec

1. Function overview The function `ui_pause()` is essentially a bash function that provides a "pause" mechanism in a shell script. It holds the execution of the script and prompts the user to press the enter key to continue. This can be particularly useful in scenarios where the user is required to acknowledge an event or review some output before the script continues executing.

2. Technical description

Name `ui_pause`

Description A simple bash function to pause the execution of a script process until the user inputs to proceed. The function uses a read command with the `-rp` option. The `-r` option prevents backslash escapes from being interpreted. The `-p` option allows the addition of a custom prompt.

Globals None

Arguments None

Outputs As standard output, it displays "Press [Enter] to continue..." and waits for the user to press the enter key.

Returns No explicit return value. The function will return the default exit status of the last command executed, in this case, `read -rp "Press [Enter] to continue..."`. If the command completes successfully, it will return `0`.

Example Usage

```
#!/bin/bash

echo "Exemplary script start"
ui_pause
echo "Exemplary script end"
```

3. Quality and security recommendations

1. It would be advisable to include validity checks on user input to prevent errors or potentially disruptive behavior. However, as the function merely expects an Enter press with no arguments, the requirement is minimal here.
2. Extend the function to customize the pause message, allowing the function to be more flexible and robust in various usage scenarios.
3. For security reasons, consider running your scripts with the principle of least privilege. This means running scripts as users with just enough permissions to perform the necessary tasks, but no more. However, as this function does not interact with system resources, it poses minimal security risk.

6.6.0.91 `ui_print_header`

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

Function signature: 57e2fede290b133fd0e31c859a63c119ad15a0eea0326adcc61d2c65983dfecc

Function overview This function, `ui_print_header()`, is a utility function designed to print a consistent header style onto the terminal for user-interface purposes. It takes one argument, a title, and outputs a three line header into the console. The header consists of a blank line, then a line of equal signs, followed by the title indented by a few spaces and finally another line of equal signs.

Technical description

- **name**: `ui_print_header()`
- **description**: This function produces a standard header on the terminal with the provided title.
- **globals**: None
- **arguments**:
  - `$1: title` - The text to be printed in header. This is usually the main title or section title of the information being printed to the terminal.
- **outputs**:
  - `(Empty Line)`
  - `====================================`
  - `$title`
  - `====================================`
- **returns**: None
- **example usage**: To use this function to print a header with title “Start of Section”, you would use `ui_print_header "Start of Section"`.

Quality and security recommendations

1. Always ensure that proper input validation is performed before accepting the title argument. This is crucial to prevent injection attacks, as malicious code may be executed if not filtered out.
2. Implement error checking for the argument. This function currently doesn’t handle cases where no argument (title) is provided.
3. It would be preferable to avoid using `echo` because it does not always handle special characters well. As a refinement, consider using `printf` or another standard output function that can handle exceptions.

6.6.0.92 `ui_prompt_text`

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

Function signature: 698fb0f6a52fc5fb7d346cb2755ab85d4e44cef66d1ac20f8f40870457b2970a

Function Overview The function `ui_prompt_text()` is a Bash function designed to provide a user-friendly text prompt in command-line interfaces. It accepts two arguments: a string to display as a prompt and a default value. If the user inputs a response, the function will return this response; otherwise, it will default to the provided second argument.

Technical Description

- **Name**: `ui_prompt_text()`
- **Description**: This function in Bash produces a text-based user prompt in a command-line interface.
- **Globals**: No global variables.
- **Arguments**:
  - `$1`: `prompt` - The text that will be displayed as the user prompt.
  - `$2`: `default` - The default value that the function will return if the user inputs no response.
- **Outputs**: Echoes the user's input back to the standard output; if no input is received, the function will output the default value.
- **Returns**: The function returns the user's input or the default value if no input is given.
- **Example Usage**:

```bash
response=$(ui_prompt_text "Enter your name" "Anonymous")
echo "Hello, $response!"
```

Quality and Security Recommendations

1. Add validation for the input parameters: Currently, the function does not validate the input parameters. It's recommended that checks be added to ensure the `prompt` and `default` arguments are provided and are strings.
2. Sanitize user input: Before processing the user input, it should be sanitized to prevent command injection or other types of attacks.
3. Error handling: The function does not handle errors, which makes it vulnerable to various types of exceptions. Incorporating error handling would enhance its stability.
4. Consider data privacy: Be aware that the user's responses could contain sensitive information, and handle and store the output with care.
5. Add documentation and comments in the code: Adding comments in the function will make the functionality clear to any other person reading or using your code.

6.6.0.93 `ui_prompt_yesno`

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

Function signature: 5b56e1af7169ad6721f24f1f595cee61d779f0b22963936d53073e284b177e9c

Function Overview The `ui_prompt_yesno` function offers a simple prompt interaction to the user. It accepts a specified question and an optional default answer. If the user does not type in a response, the function will proceed with the default answer. The function uses a while loop to keep prompting the user for input until it gets a valid response. If the user enters "y" or "n", the function will stop prompting and return with 0 for "y" and 1 for "n".

Technical Description **Name**: `ui_prompt_yesno`

Description: Function to query user response in a Y/N prompt fashion with an optional default answer.

Globals: None

Arguments: - `$1`: The prompt to display to the user. It should be a String. - `$2`: (Optional) The default answer if a user does not provide any response. Default is 'y'.

Outputs: The prompt asking for user input, with the optional default value.

Returns: - `0` if the user inputs 'y' or 'Y' - `1` if the user inputs 'n' or 'N'

Example Usage:

```
ui_prompt_yesno "Do you want to quit?" n
```

Quality and Security Recommendations

1. To avoid possible code injection, ensure that the input to `$1` and `$2` are not externally specified without proper validation or sanitizing.
2. Provide thorough comments and maintain readability to help future developers understand the code. In Bash, it's easy to write very complex one-liners, but it can be very challenging to read back and understand.
3. Error-checking methods should be adopted to handle invalid user inputs outside the bounds of 'y' or 'n'
4. Use the `-i` flag with `read` command. This makes it easier for the user to see what the default value is and perhaps only slightly modify it.

6.6.0.94 `unmount_distro_iso`

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

Function signature: 8736c022aa1702dffb2efe53cc382894bbe29e4a44b302d6fe6c2f67439871c6

Function overview The function `unmount_distro_iso` is used to unmount a given Linux distribution's ISO file. The function accepts a distribution string and uses it to create a mount point path which is then unmounted if it is currently mounted.

Technical description

- **name**: unmount_distro_iso
- **description**: This function unmounts the ISO file of a specified Linux distribution if it is currently mounted. If it is not mounted, the function just logs information about it not being mounted and returns 0.
- **globals**: [HPS_DISTROS_DIR: This is a directory path where the distribution is located]
- **arguments**: [$1: Distribution string used to create the mount point path, No other arguments]
- **outputs**: Logs info on whether the ISO was successfully unmounted or if it failed.
- **returns**:
  - 0 if the mount point is not currently mounted or if the ISO file was unmounted successfully.
  - 1 if it failed to unmount the ISO file.
- **example usage**: `bash      unmount_distro_iso Ubuntu`

Quality and security recommendations

1. Always validate inputs: It might be possible to pass in arguments that could lead to unmounting of directories that were not intended. Validate the provided distribution string to avoid these scenarios.
2. Error Handling: The error handling currently only logs the failed unmount but the script continues regardless. It might be danger in some cases to continue with the script if an unmount fails, depending on the script.
3. Redirection: Redirection of stdout and stderr may hide valuable debugging information in the event of unexpected errors, consider logging them to a file instead.
4. Process handling: It might be helpful to handler the case where the `umount` command may be stuck due to a busy device or a stale file handle.

6.6.0.95 `update_distro_iso`

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

Function signature: 2cc9510e14e685d9bd46ee12b37cf92d19a872fbfd40ed5f9884b79c76dc02d6

Function Overview The function `update_distro_iso()` is a Bash shell function designed to manage the unmounting, update and re-mounting of an ISO file that corresponds to a distribution system. The function will ask the user to manually update the ISO file before re-mounting it. The main use case for this function is when an update is required on the ISO file.

Technical Description

- Name: `update_distro_iso`

- Description: This function unmounts an ISO file related to a particular distribution system, allows the user to manually update the ISO file, then it re-mounts the updated ISO file back to its original mount point.
- Globals: [`HPS_DISTROS_DIR`: This variable stores the path of the directory where the ISO file to update is located.]
- Arguments:
  - `$1`: `DISTRO_STRING` - This argument corresponds to a string representing the particular distribution system.
- Outputs: This function outputs various statuses and instructions to the user in the terminal.
- Returns: Returns `1` if there is an error at any point in the function: either if the `DISTRO_STRING` is empty, if the unmounting fails, if the mount point is still in use after unmounting, if the ISO file is not found, or if re-mounting fails.
- Example usage: `update_distro_iso <CPU>-<MFR>-<OSNAME>-<OSVER>`

Quality and Security Recommendations

1. Fork the process that unmounts and then re-mounts the ISO to prevent potential lock-up scenarios if the user doesn't follow through with the manual update.
2. Include error checking after the re-mount operation to ensure that mounting was successful before proceeding.
3. Implement logging of function actions to help debug any potential future issues.
4. Consider encapsulating user prompts to make them more robust and prevent incorrect user input.
5. Review the global variable `HPS_DISTROS_DIR` to ensure that it properly restricts access to required directories only.

6.6.0.96 `url_decode`

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

Function signature: 51fe980cad13bbd78ebda8f1e41edbe1e1354f0ad8e6c0d8a01c7f423b453f44

Function overview The function `url_decode()` is designed to replace URL-encoded values with their original character representation. It first swaps the '+' signs with spaces, then replaces each '%xx' (where 'xx' are hexadecimal values) with their corresponding characters. This function is then used to decode a message and log it, using the logger utility. If possible, the function also writes the decoded message with associated metadata to a specified log file.

Technical description

- **Name:** url_decode

- **Description:** This function replaces URL-encoded values in a string with their original character representation. It also handles logging the decoded message via the logger utility and to a specified log file if possible. If the log file cannot be written to, it logs an error message.
- **Globals:** None.
- **Arguments:**
  - $1: This is the URL-encoded data to be decoded.
- **Outputs:** The function outputs the decoded message to stdout.
- **Returns:** This function doesn't explicitly return a value.
- **Example usage:**

  ```
  local raw_msg="%68%65%6C%6C%6F%2B%77%6F%72%6C%64"
  local msg
  msg="$(url_decode "$raw_msg")"
  echo $msg  # output: "hello world"
  ```

Quality and security recommendations

1. Always validate the inputs before operating on them. In this case, it would be wise to ensure the URL-encoded string only contains valid characters.
2. If possible, try to avoid using global variables as they may lead to unexpected behavior due to their scope. In this case, there are no global variables used which is a good practice.
3. Make sure to handle error scenarios gracefully. In this function, there's already handling when it cannot write to a log file though more checks could be added for other possible fails.
4. Ensure that the permissions of the log file are set accordingly so that only those authorized can read or write to it. This could help prevent unauthorized access to potentially sensitive information being logged.
5. Consider enhancing the function to disable logging or to log to a different location based on a config or environment variable. This enables more flexibility without changing the code.

6.6.0.97 `urlencode`

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

Function signature: f758d39e7a343eef82fc4e92ae1358118cf9a79d8accf9f5013313b5448282ac

Function Overview The `urlencode` function is a Bash function designed to encode a string in the x-www-form-urlencoded representation. This is used when encoding a string to be included in a URL. The function iterates over each character in the provided string. If the character is an alphanumeric or a small set of special characters, it's added to the output as is. Any other character is converted to its hexadecimal ASCII representation and prepended with a '%', in line with the specifications for URL encoding.

Technical Description

- **Name:** `urlencode`
- **Description:** This function is used to encode a string to be used in a URL. It operates by iterating over each character in the input string and either leaving it as is, if it's alphanumeric or a limited set of special characters, or converting it to its hexadecimal ASCII representation, prefaced by a '%'.
- **Globals:** None
- **Arguments:**
  - `$1: The string to encode`
- **Outputs:** The URL-encoded version of the input string
- **Returns:** It does not return a value. It prints the encoded string directly.
- **Example Usage:**

```
urlencode "Hello World! This needs to be encoded."
```

Quality and Security recommendations

1. Make sure to properly escape any characters when running this function to avoid command injection attacks.
2. Avoid using utf-8 characters as bash printf might not handle them well, leading to incorrect results.
3. Consider wrapping the functionality in a script with error handling to avoid potential issues with special characters in the input string.
4. Be wary of null bytes in the input as bash string operations are undefined in these scenarios.

6.6.0.98 `verify_checksum_signature`

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

Function signature: 6927fb684cf8e6a1fa690734435cbce8b702cdeef1affb7a199413ba1ed2e406

Function overview The `verify_checksum_signature` is a Bash function that verifies the checksum signature of an ISO file. It checks whether the ISO exists, fetches the checksum file and its signature online, imports the GPG key needed for verification, and then verifies the signature. Finally, it compares the ISO's checksum with the fetched checksum. If the operation successful, it returns 0 and the temp directory is removed; if there is a mismatch or an error occurs, it returns 1 and prints an informative error message.

Technical description Function:

`verify_checksum_signature`

Details:

- **Name:** verify_checksum_signature
- **Description:** Bash function that verifies the checksum signature of an ISO file.
- **Globals:**
  - HPS_DISTROS_DIR: Default directory where ISO files are stored.
- **Arguments:**
  - $1: The target CPU architecture (cpu).
  - $2: The manufacturer (mfr).
  - $3: The name of the operating system (osname).
  - $4: The version of the operating system (osver).
- **Outputs:** Error messages on failure and status messages on success. Delivered through stderr and stdout respectively.
- **Returns:** '0' if the verification is successful, '1' if an error occurs or checksum verification fails.
- **Example Usage:**

```
verify_checksum_signature $cpu $mfr $osname $osver
```

Quality and Security recommendations

1. Extend the function to support more ISO types rather than just `rockylinux`.
2. Use absolute paths in place of relative paths to guard against path injection vulnerabilities.
3. Check the availability of required utilities (curl, gpg, sha256sum, awk, etc.) at the beginning of the function.
4. Enhance error handling to provide more informative/debuggable error messages.
5. Implement a fallback mechanism for connection failures or temporary unavailability of remote resources.
6. Add a confirmation step before deleting the temporary directory.
7. Test this function with multiple edge-cases and unexpected inputs to ensure reliability.

6.6.0.99 `verify_required_repo_packages`

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

Function signature: a16392408f8b201975834d3d38029e3859302d45f825b35156dca0ae85e71609

Function overview The function `verify_required_repo_packages` checks the existence of required software packages in a specified repository directory. It takes the path of the repository and an array of needed packages as input. If repository path does not exist or isn't specified or the required package(s) do not exist in the specified directory, it logs an error message and returns a non-zero status. Otherwise, it logs a success message stating all required packages are present, and also returns zero.

Technical description

- **Name:** verify_required_repo_packages
- **Description:** Verify whether specified packages are present in given repository directory. Log error messages for missing repository path or required packages and return non-zero status in such cases.
- **Globals:** None
- **Arguments:**
  - `$1: repo_path` - The fully qualified path to the repository containing the packages
  - `$@: required_packages` - After shifting, this variable contains the array of required package names
- **Outputs:** Logs error or info messages about the availability of required packages
- **Returns:**
  - `0` if all required packages are present
  - `1` if the repository path is not provided or does not exist
  - `2` if one or more required packages are missing
- **Example usage:** `bash verify_required_repo_packages "${HPS_PACKAGES_DIR}/${DIST_STRING}/Repo" zfs opensvc`

Quality and security recommendations

1. Implement robust error-handling: Potentially extract finer-grained error codes for diverse problematic situations such as repository directory not being accessible, or the directory contains no packages.
2. Add input validation: Enhance the function with additional checks for the input parameters. Check for appropriate formats and values of the input parameters.
3. Portability: Ensure that the function behaves as expected across different systems or different versions of bash.
4. Security: Make sure that appropriate permissions are set for directory access and also ensure it can't be manipulated to cause a potential security issue.
5. Performance: Consider using more efficient bash constructs to increase the performance of the function. For example, instead of using a loop to check each package, consider using associative arrays if the number of needed packages is large.

6.6.0.100 `verify_rocky_checksum_signature`

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

Function signature: 63d87412289beb590b23dead3edfbcd2afb85d4ee141526d80e8ad9104dcfbcb

Function Overview The following function, `verify_rocky_checksum_signature`, is designed to download and validate the checksum of Rocky Linux iso files by leveraging the accompanying gpg signature. The function performs several actions: it first downloads the checksum and the corresponding signature. It then imports the official Rocky Linux GPG key. Afterward, it verifies the signature against the checksum file. Finally, it extracts the expected checksum from the file and matches it to the actual checksum of the iso. If the checksum are identical, the function will signify successful verification.

Technical Description

- Name: `verify_rocky_checksum_signature`
- Description: This function primarily aims to download and authenticate the Rocky Linux iso files via the associated gpg signature.
- Globals: [HPS_DISTROS_DIR: The directory where the distributions are stored]
- Arguments: [$1: Specifies the version of Rocky Linux]
- Outputs: Status messages regarding the actions performed (downloading checksum and signature, importing GPG keys, verifying checksum)
- Returns: 0 if GPG signature and checksum are both verified successfully, otherwise returns 1, 2, 3 or 4 depending on the specific error
- Example usage:

```
verify_rocky_checksum_signature "8.5"
```

1. Make proper use of quoting: Variables should always be contained within double quotes in case they contain special characters (for example, spaces).
2. Ensure you are relying on trusted sources: The function downloads scripts from the internet, but always be sure you trust the source before running a script.
3. Check for error conditions: The function does check for error conditions and reports results, which is good.
4. Avoid suppressing errors: The use of `2>/dev/null` suppresses important error messages. A better practice is to handle all possible exceptions or errors.
5. Consider adding more comments: Useful comments can help others understand the purpose and function of your code.
6. Handle command failures: The function should stop executing as soon as any of the commands fails. This will prevent the condition where a failure leaves the system in an indeterminate state. Use `set -e` or check the status of each command manually.
7. Use updated cryptographic standards: Always use updated secure algorithms (like SHA-256) during verification.

6.6.0.101 `write_cluster_config`

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

Function signature: cf6f48116ee4f6ae2fa075ed74a4476f6c22ffe594564ebf704d40d3dce09039

1. Function overview The function `write_cluster_config` is designed to write a set of provided values into a specified target file as a cluster configuration. If no values are supplied, the function will return an error message indicating that an empty cluster configuration cannot be written. If values are provided, they will be written to the target file and a confirmation message will be displayed.

2. Technical description

- **Name**: `write_cluster_config`
- **Description**: The `write_cluster_config` function takes a filename and a series of values as arguments. It writes these values into the given filename. If no values are provided, the function will display an error message and halt operation.
- **Globals**: None.
- **Arguments**:
  - `$1`: The target file into which the configuration values should be written.
  - `$@`: The values, passed as an array, that should be written into the cluster configuration.
- **Outputs**: An error message if no values are provided, a logging message of what is being written, and a success message upon successful write.
- **Returns**: 1 if no values are provided for writing, otherwise no explicit return.
- **Example usage**: Assume an array `arr=("value1" "value2" "value3")`. We can call `write_cluster_config "target.txt" "${arr[@]}"`

3. Quality and security recommendations

1. Escape the output using secure methods to prevent possible command injection vulnerabilities or unexpected behavior.
2. Implement error handling if the target_file does not exist or cannot be written to.
3. Add checks to ensure the validity of the values to be written to the file.
4. Implement logging system to keep track of the function operations.
5. Utilize secure temporary files during operations to avoid inadvertent exposure of potentially sensitive data.
6. Add testing routines to ensure the function behaves as expected under a variety of conditions.

7 Reference

Static technical reference information for HPS, including function documentation, troubleshooting guides, and configuration details.

7.1 Environment variables

Stub: Explanation of exported variables and their purpose.

7.2 Glossary

Stub: Definitions of acronyms and technical terms.

7.3 Reference configurations

Stub: Example `cluster.conf`, `host.conf`, and service configuration files.

7.4 Troubleshooting

Stub: Common problems, causes, and fixes for HPS operation.