OPEN DIGITAL
CONSULTING CO

Documentation

HOX project

Stuart J Mackintosh

Wednesday 30 July 2025

Contents

1 How to Write Documentation for Bash Functions

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

1.1 Best Practice: Function Documentation Format

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

Template:

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

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

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

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

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

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

Example:

```
# greet_user: Prints a personalized greeting.

# Globals:
```

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

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

# Outputs:
# Writes greeting to STDOUT.

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

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

1.2 Key Points

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

1.3 Additional Best Practices

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

1.4 Why This Matters

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

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

2 Function documentation

2.1 build_yum_repo

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

2.1.1 Function overview

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

2.1.2 Technical description

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

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

2.1.3 Quality and security recommendations

1. Consider adding more comments within the function to increase readability.

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

2.2 check_and_download_latest_rocky

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

2.2.1 Function Overview

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

2.2.2 Technical Description

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

```
check_and_download_latest_rocky
```

2.2.3 Quality and Security Recommendations

- Always validate the URL before using cURL for downloads.
- Implement error handling for failed cURL downloads and directory creations.

- Check if the global variable `HPS_DISTROS_DIR` is set before the function is called.
- Consider adding an argument to specify the architecture or ISO type gaining more flexibility.
- For security, consider verifying the checksum of the downloaded ISO to ensure it is not tampered with.

2.3 `check_latest_version`

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

2.3.1 Function overview

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

2.3.2 Technical description

- name: `check_latest_version()`
- description: The function checks for the latest version of an operating system.
- globals: None
- arguments:
  - `$1: cpu` - the type of cpu
  - `$2: mfr` - the manufacturer
  - `$3: osname` - the name of the operating system
- outputs: Echoes the status of version checking, any potential error messages, and the latest version number if found.
- returns: Returns 1 when an error occurs, or 0 on successful finding of the latest version.
- example usage:

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

2.3.3 Quality and security Recommendations

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

2.4 `download_iso`

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

2.4.1 Function Overview

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

2.4.2 Technical Description

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

  - `0` if the target ISO file already exists or if the download is successful.
  - `1` if an unsupported OS variant is entered or if downloading the ISO file fails.
- **Example Usage**:

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

2.4.3 Quality and Security Recommendations

1. Add more comprehensive error handling and messaging for different possible failure points (e.g. unsupported CPU types, unavailable URL, network errors).
2. Instead of hardcoding a base URL, consider an external configuration file or environment variables.
3. Restrict the rights of the target directory and the downloaded ISO files to limit potential security risks.
4. Consider additional verifications after the download. (e.g., checksum verification to ensure the integrity of the downloaded file)
5. Include comments describing each part of the function, making it easier for others to understand how the function works.

2.5 `extract_iso_for_pxe`

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

2.5.1 Function overview

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

2.5.2 Technical description

Name: `extract_iso_for_pxe`

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

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

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

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

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

Example usage:

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

2.5.3 Quality and security recommendations

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

2.6 `extract_rocky_iso_for_pxe`

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

2.6.1 Function overview

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

2.6.2 Technical description

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

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

2.6.3 Quality and security recommendations

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

2.7 `fetch_and_register_source_file`

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

2.7.1 Function overview

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

2.7.2 Technical description

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

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

2.7.3 Quality and security recommendations

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

2.8 fetch_source_file

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

2.8.1 Function overview

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

2.8.2 Technical description

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

2.8.3 Quality and security recommendations

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

2.9 generate_dhcp_range_simple

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

2.9.1 Function overview

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

2.9.2 Technical description

Function: `generate_dhcp_range_simple()`

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

2.9.3 Quality and security recommendations

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

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

2.10 get_external_package_to_repo

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

2.10.1 Function Overview

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

2.10.2 Technical Description

Name: `get_external_package_to_repo()`

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

Globals: None

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

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

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

Example Usage:

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

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

2.10.3 Quality and Security Recommendations

1. Implement validation to ensure that the URL is a HTTPS URL to provide an added layer of security.
2. Implement checksum validation to ensure the integrity of the downloaded file.

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

2.11 get_iso_path

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

2.11.1 1. Function overview

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

2.11.2 2. Technical description

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

- **Name**: `get_iso_path`
- **Description**: This function checks if the `HPS_DISTROS_DIR` variable is set and whether it indicates a valid directory. If so, it appends "/iso" to it and returns the resulting string. If not, it raises an error and returns the exit code 1.
- **Globals**: [`HPS_DISTROS_DIR`: The directory where the ISO files are stored.]
- **Arguments**: [None]
- **Outputs**: If successful, it outputs the path to an ISO file. If it fails, it sends an error message to stderr.
- **Returns**: It returns 0 if successful, otherwise it returns 1.
- **Example usage**: Not applicable, as the function does not take any arguments. It can be called in a script as `get_iso_path`.

2.11.3 3. Quality and security recommendations

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

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

2.12 has_sch_host

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

2.12.1 Function Overview

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

2.12.2 Technical Description

- Name: `has_sch_host`
- Description: This function scans through the configuration directory for `.conf` files and checks if there is at least one file of type `'SCH'`. If such a file exists, the function returns success, otherwise it returns failure. If the configuration directory does not exist, an error message is output and the function returns failure.
- Globals:
  - `HPS_HOST_CONFIG_DIR`: This global variable is used to define the path of the host configuration directory that is to be searched.
- Arguments: None
- Outputs: Error message if the host config directory specified by `HPS_HOST_CONFIG_DIR` is not found.

- Returns:
  - `0` if at least one SCH type config file is found.
  - `1` if no SCH type config file is found or if the config directory is not found.
- Example Usage:

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

2.12.3 Quality and Security Recommendations

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

2.13 host_config_delete

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

2.13.1 Function overview

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

2.13.2 Technical description

- **Name:** `host_config_delete()`
- **Description:** The function deletes a host configuration file based on the provided MAC address.
- **Globals:** [HPS_HOST_CONFIG_DIR: This global is used to specify the directory of the host configuration files.]

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

2.13.3 Quality and security recommendations

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

2.14 `host_config_exists`

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

2.14.1 Function overview

The `host_config_exists()` function is a BASH script utility designed to check whether specific host configuration files exist in a designated directory. It takes a Media Access Control (MAC) address as the argument, forms a path to the should-be existing file, and checks if the .conf file indeed exists at the specified location.

2.14.2 Technical description

- **Name**: `host_config_exists`
- **Description**: This function checks if a configuration file named after the provided MAC address exists in the predefined host configuration directory.
- **Globals**: `HPS_HOST_CONFIG_DIR`: The directory where host configuration files are stored.
- **Arguments**:
  - `$1`: MAC address of the host whose configuration file's existence will be checked
- **Outputs**: None, the function doesn't produce any output.

- **Returns**: An exit statuses representing 'true' (if file exists) or 'false' (if file does not exist).
- **Example usage**:

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

2.14.3 Quality and security recommendations

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

2.15 host_config

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

2.15.1 Function Overview

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

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

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

2.15.2 Technical Description

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

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

2.15.3 Quality and Security Recommendations

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

2.16 `host_config_show`

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

2.16.1 Function overview

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

it by trimming and escaping special characters in the value associated with each key. After processing, it then echoes these key-value pairs.

2.16.2 Technical description

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

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

2.16.3 Quality and security recommendations

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

2.17 `host_initialise_config`

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

2.17.1 Function overview

The function `host_initialise_config` is primarily used for setting up the host configuration. This function takes a local MAC address as an argument, generates a

configuration file name using that MAC address and the host configuration directory `${HPS_HOST_CONFIG_DIR}`. It then creates the mentioned directory, sets the state to "UNCONFIGURED", and logs the completion of the host config initialization process.

2.17.2 Technical description

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

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

2.17.3 Quality and security recommendations

1. Always ensure that the MAC address argument is valid or sanitized before passing it to the function as it may lead to unhandled exceptions.
2. Add error checking logic and include exception handling for actions like non-existent directories or inaccessible file paths.
3. Ensure that the use of `hps_log` is secure and does not expose sensitive logs in insecure locations.
4. Uncomment the pieces of code related to the creation timestamp and refine in a way that doesn't cause an error. This will enhance traceability and aid in debug processes.
5. Explore mechanisms to return status or error codes to make the function more robust and usable in the script. The function currently does not return any value.
6. Consider encrypting sensitive data in the configuration file to enhance security.

2.18 `host_network_configure`

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

2.18.1 Function Overview

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

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

2.18.2 Technical Description

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

2.18.3 Quality and Security Recommendations

- The function should return unambiguous values consistently. It does not explicitly specify a return on success, just failure.
- The function utilizes the `ipcalc` command, the absence of which triggers a non-zero exit code. This creates a dependency which should be clearly documented.
- Integrate input validation to ensure that the `macid` and `hosttype` arguments being passed to the function are well-formed to avoid undefined behavior.
- Log sensitivity data such as IP addresses, CIDR notations, netmasks etc. can lead to security risks. Always ensure that logs do not expose sensitive details unless absolutely necessary and approved.
- Employ permission checks to ensure that the script is not executed with higher than necessary privileges, which can potentially compromise security.

- The function should include more detailed error handling, including potential issues with sourcing `cluster_config`, and extraction of `dhcp_ip` and `dhcp_cidr`.

2.19 `int_to_ip`

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

2.19.1 Function Overview

The `int_to_ip()` function is used to convert an integer into the corresponding IP address. The function does this through bitwise shifting and AND operation with 255. The function is used as part of a larger script, where it is used to check for available IP addresses in the network base and assign one of them to a new host. The script keeps checking for available IP addresses and hostnames from two designated lists and assigns the ones that are not yet used to the new host. Defaults and any other settings for the new host are set in the script as well.

2.19.2 Technical Description

- name: `int_to_ip`
- description: The `int_to_ip` function converts an integer into a corresponding IP address.
- globals: [base_int: The integer value of an IP address, HPS_HOST_CONFIG_DIR: The directory where host config files are stored, max: The upper limit when checking for available IP addresses]
- arguments: [$1: This is the integer which will be converted into an IP address]
- outputs: Prints the corresponding IP address of the given integer on the standard output
- returns: Does not return anything
- example usage: `ip_address=$(int_to_ip $integer)`

2.19.3 Quality and Security Recommendations

- Always check the validity of the integer before using it as an argument for the `int_to_ip` function.
- Secure your bash scripting with good practices such as strict mode, by adding `set -euo pipefail` at the top of your scripts. This will exit the script if any command, pipeline or substitution fails.
- Always use “$()” for command substitution instead of backticks ““”.
- Use [[instead of [for tests.
- Quote all variables to prevent word splitting.

- Add more comments in the code to make it more understandable and maintainable.
- Implement checks to ensure that HPS_HOST_CONFIG_DIR directory exists and is accessible.
- Make sure that `host_config` and `hps_log` functions do proper error checking.
- Consider using more descriptive variable names for better code readability.

2.20 ip_to_int

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

2.20.1 1. Function Overview

The `ip_to_int()` function is a Bash function that takes an IP address as a single argument and returns the equivalent integer representation. It first splits the IP address into four octets, and then performs an arithmetic operation to convert the four octets into a single integer. This type of representation is commonly used in networking configurations and calculations.

2.20.2 2. Technical Description

- **Name:** `ip_to_int`
- **Description:** This function converts an IPv4 address into its integer representation.
- **Globals:** None
- **Arguments:**
  - `$1`: An IPv4 address string in the form of 'xxx.xxx.xxx.xxx'.
- **Outputs:** This function outputs the integer representation of the argument IP address.
- **Returns:** It does not explicitly return any value, but it echoes the output to stdout (which could be captured by callers as required).
- **Example Usage:**

```
ip_to_int 192.168.0.1
```

2.20.3 3. Quality and Security Recommendations

- Use local variables inside the function to avoid overwriting any potential global variables named `o1`, `o2`, `o3`, or `o4`.
- Perform validation on the input IP address before attempting to convert it, to ensure it is well-formed and contains 4 octets.

- Use `printf` instead of `echo` for output, as it is safer and more predictable. `echo` can behave unexpectedly with certain inputs.
- The script has no return error handling; it implicitly assumes the input will be valid. A robust function would include error checking and handling.
- `IFS` environment variable is changed without being reset; it may affect subsequent code depending on its value. It's better to saving its old value and restore it at the end of the function.

2.21 `list_local_iso`

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

2.21.1 Function overview

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

2.21.2 Technical description

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

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

2.21.3 Quality and security recommendations

1. Validate the input parameters.
2. Add error handlers for potential issues - for example, what happens if the directory specified by `get_iso_path` does not exist.
3. Make sure that `get_iso_path` provides a correct path to prevent potential path traversal attacks.

4. Output all echo messages not only to the standard output but also to a dedicated log file with timestamps for better tracking.
5. Make sure the function works as expected with special characters in the `cpu`, `mfr`, `osname`, and `osver` parameters.
6. Implement an exit mechanism to break the loop if it runs for a certain amount of time to avoid potential infinite looping.

2.22 mount_distro_iso

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

2.22.1 Function Overview

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

2.22.2 Technical Description

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

2.22.3 Quality and Security Recommendations

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

2.23 `prepare_custom_repo_for_distro`

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

2.23.1 Function overview

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

2.23.2 Technical description

Name: prepare_custom_repo_for_distro

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

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

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

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

Returns: - Returns 0 if the repository was successfully prepared - Returns 1 if the directory creating process fails - Returns 2 if the downloading process fails - Returns 3 if the file copy

process fails - Returns 4 if the package source is invalid - Returns 5 if the metadata creating process fails - Returns 6 if any required packages are missing

Example Usage:

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

2.23.3 Quality and security recommendations

1. Enforce stricter validation of input. This includes vetting the provided URLs for malicious content.
2. Handle file permission correctly when creating new directories and files.
3. Consider catching errors with specific error handling functions.
4. Use checksums or cryptographic hashes(like SHA256) to ensure the integrity of downloaded packages.
5. Check if there are redundant process which can be eliminated or optimized.
6. Log more detailed information for debugging in case of error.
7. Implement testing to catch and find bugs before production.

2.24 register_source_file

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

2.24.1 Function overview

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

2.24.2 Technical description

- **name**: `register_source_file`
- **description**: A Bash function that checks and registers a filename and handler as a source file in an index file found in the HPS_PACKAGES_DIR directory, or the default directory if not set. It prevents registering duplicate entries.
- **globals**: [HPS_PACKAGES_DIR: A variable for the target directory where the source files are registered. If not set, the default directory is `/srv/hps-resources/packages/src`.]
- **arguments**: [$1: Filename to be registered,$2: Handler for the file]

- **outputs**: Echoes the action (registration or lack thereof due to duplicates) and the file and handler details.
- **returns**: 0 (when the file is already registered)
- **example usage**: `register_source_file "newfile.sh" "myhandler"`

2.24.3 Quality and security recommendations

1. Consider adding checks for the validation of arguments provided. For instance, ensuring that they are not null or verifying their format improves robustness.
2. Include error handling for the cases when the target directory cannot be created. This will be helpful for diagnosing potential issues.
3. Using absolute path names can enhance the script's security to avoid any potential relative-path attacks.
4. Inputs should be sanitized to prevent possible injection attacks. For example, if the handler is not securely sanitized, an attacker might manipulate it to execute arbitrary code.
5. Always use the double quotes around variable interpolations to prevent word splitting and glob expansion.
6. Implement clear and detailed logging. In addition to the operation result, these logs could include timestamps, user IDs, and more granular details about the operations. This improves traceability and debugging in case of unexpected behavior or errors.

2.25 `rocky_latest_version`

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

2.25.1 Function overview

This bash function, `rocky_latest_version`, fetches the page content from the Rocky Linux download page and parses it to find the latest version number of Rocky Linux available for download. The function leverages `curl`, `grep`, `sed`, and `sort` tools to fetch and parse HTML content, and extract version numbers from it. If successful, it prints the most recent version number to the standard output.

2.25.2 Technical description

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

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

2.25.3 Quality and security recommendations

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

2.26 `unmount_distro_iso`

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

2.26.1 Function overview

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

2.26.2 Technical description

Name: `unmount_distro_iso`

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

Globals: - `HPS_DISTROS_DIR`: Directory path of distributions

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

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

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

Example usage: `unmount_distro_iso ubuntu-20`

2.26.3 Quality and security recommendations

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

2.27 update_distro_iso

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

2.27.1 Function overview

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

2.27.2 Technical description

- **name**: `update_distro_iso`
- **description**: A Bash function that handles the unmounting, updating, and re-mounting of a Linux distribution ISO file.
- **globals**: `[ HPS_DISTROS_DIR: A global variable that holds the root directory for multiple Linux distributions. ]`
- **arguments**: `[ $1: Distinctive string representing a Linux distribution, typically in the format <CPU>-<MFR>-<OSNAME>-<OSVER> ]`

- **outputs**: Outputs to stdout, mainly informing the user about the status of the ISO file, like whether it is mounted/unmounted, or whether the user can update the ISO file or not.
- **returns**: Returns 1 if the DISTRO_STRING is not provided, the iso-file of the given distribution is not found or mounting or unmounting fails.
- **example usage**:

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

2.27.3 Quality and security recommendations

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

2.28 `urlencode`

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

2.28.1 Technical description

- Name: urlencode
- Description: This function converts any given string into a URL-encoded string. It does this by converting special characters in a string to their equivalent hexadecimal escape sequences.
- Globals: [None]
- Arguments:
  - $1: The string that needs to be URL-encoded.
- Outputs: The function outputs a URL-encoded string.
- Returns: None.
- Example Usage:

```
urlencode "Hello World!"
# Returns: Hello%20World%21
```

2.28.2 Function overview

The `urlencode` function takes a string as an input and converts it into a URL-encoded string by iterating over each character in the string. For each character that is not alphanumeric or a character from the list (.~-*), it is converted to a hexadecimal representation and prepended with a '%'. Alphanumeric and (.*~-) characters are copied as-is. The URL-encoded string is then printed out.

2.28.3 Quality and security recommendations

Here are some suggested improvements:

- Consider implementing error handling: The function currently does not handle errors that could potentially occur during the execution such as unsuccessful conversions. It might be useful to add some sort of error handling mechanism.
- Make sure inputs are sanitized: To protect against possible injection attacks, the provided input should be validated and sanitized where necessary.
- Enhance readability: While the function is compact, it can be made more readable. Especially, the case statement handling the conversion can be better named.
- Code review and testing: Always have your code reviewed by peers and make sure to write tests to cover all possible corner cases. This makes sure your code is both effective and efficient.

2.29 `verify_checksum_signature`

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

2.29.1 Function overview

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

2.29.2 Technical description

- **Name**: `verify_checksum_signature`

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

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

2.29.3 Quality and security recommendations

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

2.30 `verify_required_repo_packages`

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

2.30.1 1. Function Overview

This function `verify_required_repo_packages()` is used in a Linux system to validate the existence of specified packages within a given software repository. The function initiates by taking in the repository path as well as the names of necessary packages as arguments. It validates the existence of the repository path and prompts an error log if the path does not exist. The function then iterates through the list of required packages, checking their existence in the repository. If any package is missing, its name

is captured in a 'missing' array. An error log is prompted if any required packages are not found in the repository. If all packages exist in the repository, a success log message is printed.

2.30.2 2. Technical Description

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

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

2.30.3 3. Quality and Security Recommendations

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

2.31 `verify_rocky_checksum_signature`

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

2.31.1 Function Overview

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

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

2.31.2 Technical Description

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

2.31.3 Quality and Security Recommendations

1. To improve security, consider using more secure encryption methods or addition of password or passphrase for GPG keys.
2. Implement a strategy for handling updating GPG keys.
3. Expand the function to handle different processor architectures other than “x86_64”.
4. In terms of quality of code, consider refactor or externalize the repeated `curl` commands into a separate function.
5. If the external resources (e.g., public keys) move or are renamed, the script will fail. Employ an error-handling mechanism for unavailable resources.
6. Validate user input to ensure it fits expected parameters. For instance, verifying version number is a positive integer.