blib Code Reference v1.4

David Hobach

Contents

blib

blib - a bash library

The basic functions which are imported by default.

version: Execute `blib version` or use b_version.

Disclaimer

the
Lesser GNU General Public License for more details.

You should have received a copy of the Lesser GNU General Public License along with this program. If not, see https://www.gnu.org/licenses/.

The above statements apply to all modules of blib if not mentioned otherwise.

Coding Conventions

1. embrace the KISS principle
2. some general guidelines: http://www.kfirlavi.com/blog/2012/11/14/defensive-bash-programming/
3. use b_[module]_[camel case function name] to denote functions meant to be used by users of the library, B_[module]_[upper case var name] for global variables
4. use BLIB_[module]_[upper case var name] for global variables (to be avoided whenever possible) not meant to be used by library users ("private" variables); library users can use setters or getters
5. use the BLIB_STORE with the above naming conventions for "private" variables whenever possible
6. use blib_[module]_[camel case function name] to denote private functions not meant to be used by library users; the function name should not contain any underscores
7. the blib module itself is the only exception which can use b_, B_, blib_, BLIB_ without module name
8. prefixes such as t_, T_ and UTD_ are exclusively related to test code
9. set exit codes (!= 0 –> issue) wherever it makes sense
10. keep the global namespace clean whenever possible
11. declare -g must be used in order to allow sourcing from a function (b_import)
12. modules must check their dependencies via b_deps
13. modules should provide a header similar to that of the blib source file in order to make the documentation generation work
14. modules may be placed in subfolders of arbitrary depth
15. use 0 to indicate true and 1 to indicate false for variables; for exit codes use a non-zero exit code to indicate the number of errors
16. functions should be tagged with the following tags, if applicable:
    @StateChanging - the function changes the internal state of the script in a way that will not propagate to supershells (e.g. global variables) and should thus not be called from subshells (unless the user wants the state to only change in that shell)
    @B_E - the function uses B_E for error handling and may thus behave differently depending on the implemented error handler

Library Usage

with the default bash options:

```
source blib
b_checkVersion 1 0 || { >&2 echo "This script depends on blib (https://github.com/3hhh/blib
b_import [module]
```

Dependencies

bats
cat
dirname
find
mktemp
mv
readlink
rm
sort
su
whoami

Imports

no imports

Global Variables

B_LIB_DIR *blib/B_LIB_DIR*
Path of the blib installation directory.

B_TEST_MODE *blib/B_TEST_MODE*
blib will set this variable to 0, if blib is running in test mode.
This variable may be used to bypass code during testing, if bats cannot test that code due to its limitations (e.g. for EXIT traps which bats uses for itself).

B_CALLER_NAME *blib/B_CALLER_NAME*
Name of the executable or script as String which called blib and any child libraries.

B_ERR *blib/B_ERR*
Global variable used for error handling throughout blib, cf. B_E.

It is recommended to always set an at least partially static error message on confirmed errors as variables may be empty (which would indicate "no error" for B_E).

B_RC *blib/B_RC*
Can be used to set the return code for B_E in the case of an error. It defaults to 1.

This should be set to an integer value between 1 and 255. Any other value may cause undefined behaviour.

B_CONF_DIR *blib/B_CONF_DIR*
Path to the blib configuration directory. Modules may create subfolders named by their module name there. May not exist.

B_SCRIPT *blib/B_SCRIPT*
Global variable which can be used to obtain the two global variables B_SCRIPT_DIR and B_SCRIPT_NAME as follows:

```
eval "$B_SCRIPT"
```

B_SCRIPT_NAME *blib/B_SCRIPT_NAME*
Path of the sourced or executed bash script executing the eval (symlinks are resolved) of B_SCRIPT.

B_SCRIPT_DIR *blib/B_SCRIPT_DIR*
Name of the sourced or executed bash script executing the eval (symlinks are resolved) of B_SCRIPT.

Global Aliases

Alias expansion is automatically enabled by blib as it is required for its core functionality. So if you have strange aliases defined in your shell environment, this may cause undefined blib behaviour.

B_E *blib/B_E*
The blib error handler: All blib modules use it whenever execution errors require special handling that the currently executing code cannot achieve.

Syntax:

```
B_ERR="This is an error message." ; B_E ;
```

If you need to set the return/exit code, you can do it with B_RC:

```
B_ERR="This is another error message." ; B_RC=6 ; B_E ;
```

Calling B_E means:
Check B_ERR for an error message and if there is one, handle it. It can be placed at the end of a line or on its own line. B_E will then process the error message in the way defined by the error handler (cf. b_defaultErrorHandler) and stop any further execution of code in the current context (function, script,

. . .) returning a non-zero exit code (1) unless the described error was fixed. In the latter unlikely case it'll let execution proceed.

More examples:

```
#handle a potentially failing command:
cmd || { B_ERR="cmd failed." ; B_E ; }

#capture stdout of a potentially failing command:
local out=
out="$(cmd)" || { B_ERR="cmd failed." ; B_E ; }

#handle multiple potentially failing commands in a try/catch like manner:
(   set -e
    cmd1
    cmd2
    cmd3
) #NOTE: || doesn't work here!
if [ $? -ne 0 ] ; then
    B_ERR="Some commands failed."
    B_E
fi
```

The error handler can be re-defined at runtime with b_setErrorHandler.

Functions

b_printStackTrace [skip level] *blib/b_printStackTrace*
[skip level]: skip that many levels of the stack trace (optional, default: 1 - skip this function call)
print the current stack trace in a human readable way
returns: stack trace with the first levels skipped as defined

b_nop *blib/b_nop*
Do nothing.
returns: nothing; sets a zero exit code

b_version [part] *blib/b_version*
Get the version of this blib instance.
[part]: Optional parameter defining the part of the version to retrieve (0: all as String (default), 1: major as Integer, 2: minor as Integer).
returns: blib version as string; always sets a zero exit code

b_checkVersion [minimum allowed major] [minimum allowed minor] [maximum allowed major] [maximum allowed minor] *blib/b_checkVersion*
Check whether the currently running blib instance meets the given blib version

requirements.
To e.g. make sure you're on blib version 1.1 or higher, use

```
source blib
b_checkVersion 1 2 || { >&2 echo "This script depends on blib (https://github.com/3hhh/blib
```

[minimum allowed major]: The lowest acceptable major version number (default: 0).
[minimum allowed minor]: The lowest acceptable minor version number (default: 0).
[maximum allowed major]: The highest acceptable major version number (default: infinite).
[maximum allowed minor]: The highest acceptable minor version number (default: infinite).
returns: Sets a zero exit code, if an only if the currently running blib instance meets the version requirements.

b_defaultErrorHandler [error out] [send err] [send stack trace]
blib/b_defaultErrorHandler
The blib default error handler.
As any error handler it must

1. handle the error message (if not the error itself) lying in B_ERR
2. not take any non-numeric arguments
3. not error out itself
4. implement the below *[error out]* as its first parameter (to make b_setBE work)
5. ideally use b_error to send error messages to the user
6. return one of the following exit codes:
   a) 0: **if and only if** the error was fixed entirely and the caller may ignore the error (i.e. probably never)
   b) 1: The error wasn't fixed. Functions should return to their caller indicating an error (non-zero status code). Direct shell calls will exit. B_ERR is **not** reset to blank, i.e. the next call to B_E in the same context will cause another error. The caller may use this to either *throw* the error further or handle and clear the error.
   c) 2: Force a stop of execution in the current shell / error out.

[error out]: Whether or not to call exit after the error message handling, if the error couldn't be handled (default: 0 = always error out / call exit). If set to 1, B_E will allow e.g. functions to return to their callers.
[send err]: Whether or not to send the error message to the user (default: 0 = send).
[send stack trace]: Whether or not to send a stack trace to the user (default: identical to [send err]).
returns: see the description above

b_setBE [error out] *blib/b_setBE*
Set the [error out] behaviour of the currently configured error handler.

Contrary to b_setErrorHandler this function may be called by blib modules as all error handlers are required to support [error out] as parameter.

Example for switching the error out behaviour:

```
b_setBE 1
funcThatMayCallB_E #without subshell
ret=$?
b_resetErrorHandler
```

[error out]: see b_defaultErrorHandler (default: 0)
returns: nothing, always sets a zero exit code
@StateChanging

b_setErrorHandler [handler] *blib/b_setErrorHandler*
Set the error handler for all future exections of B_E in the current scope.

You can do this in e.g. subshells to limit the effect.

blib modules should only use this function if absolutely necessary to temporarily modify the error behaviour whilst making sure that b_resetErrorHandler is called in the end. Otherwise it will prevent library users from setting the general behaviour in their scripts.

Usually you do not want to write an entirely new handler, but modify the b_defaultErrorHandler parameters with this setter or use b_setBE for that.
[handler]: Function to handle errors. See b_defaultErrorHandler for details.
returns: nothing
@StateChanging

b_resetErrorHandler [reset B_ERR] *blib/b_resetErrorHandler*
Set the error handler to whatever it was before the last call to b_setErrorHandler or b_setBE.
[reset B_ERR]: Whether or not to also reset B_ERR (default: true/0).
returns: nothing, always sets a zero exit code
@StateChanging

b_getErrorHandler *blib/b_getErrorHandler*
Get the currently for B_E configured error handler.
returns: the error handler function

b_silence [function] [param 1] .. [param p] *blib/b_silence*
Call the given function with its parameters in the current shell context whilst suppressing all of its output to both stdout and stderr. Anything written to B_ERR however is passed to B_E (which can still write to stderr).

This function is useful when you want to keep an error message set with B_ERR, but discard everything else.
In contrast `yourfunction &> /dev/null` may also drop the error message, if you're using an error handler (see B_E) that writes to stdout or stderr.
[function]: The function to execute.
[param p]: An arbitrary number of function parameters.
returns: Sets the status code of the called function, but doesn't print anything. B_E is called on errors.
@B_E

b_assertLastPipes [error message] *blib/b_assertLastPipes*
Assert that the last pipe statements all had a zero exit code.
An alternative to `set -o pipefail`.
[error message]: Optional error message to use for B_E in case the assertion fails.
returns: Nothing. Assertion failures trigger B_E.
@B_E

b_defaultMessageHandler [message type] [message] [first part] [last part] *blib/b_defaultMessageHandler*
Handles the given message by printing information to stdout and errors to stderr. This is the default message handler used by blib. It can be changed by b_setMessageHandler.
Message handler implementations must support at least the parameters of this function, but may add additional parameters. If an handler implementation does not support partial messages (i.e. the [first part] and [last part] parameters), it should be chained to a b_cachingMessageHandler.
[message type]: 0=informational message, 1=error message.
[message]: String representing the message.
[first part]: If set to 0, assume that the given message is the first part of an overall chain of messages (default: 0).
[last part]: If set to 0, assume that the given message is the final part of an overall chain of messages (default: 0).
returns: Nothing. Never causes errors.

b_cachingMessageHandler [message handler] [message type] [message] [first part] [last part] *blib/b_cachingMessageHandler*
A message handler implementation which caches partial messages for a receiving message handler that cannot handle them until they are completed.
Use `b_initCachingMessageHandler && b_setMessageHandler "b_cachingMessageHandler [your messag` to set your receiving message handler as message handler to cache messages for.
This implementation uses the b_getDefaultMessageHandlerIntermediate as separator between partial messages.
[message handler]: The function implementing the receiving message handler. It must handle the [message type] and the [message].

[message type]: See b_defaultMessageHandler.
[message]: See b_defaultMessageHandler.
[first part]: See b_defaultMessageHandler.
[last part]: See b_defaultMessageHandler.
returns: Nothing. Never causes errors.

b_initCachingMessageHandler [maximum message type] *blib/b_initCachingMessageHandler*
Initializes a new cache for a b_cachingMessageHandler. This function *must* be called at least once before using b_cachingMessageHandler.
[maximum message type]: Maximum message type for which to allocate caches (default: 1 –> allocate caches for 0..1/info..error).
returns: Nothing.
@B_E

b_setDefaultMessageHandlerIntermediate [intermediate] *blib/b_setDefaultMessageHandlerIntermedia*
[intermediate]: String to use as intermediate between two message parts with b_defaultMessageHandler, i.e. the resulting message should be [part 1][intermediate][part 2]. Default: Space.
returns: Nothing.

b_getDefaultMessageHandlerIntermediate *blib/b_getDefaultMessageHandlerIntermediate*
Get the currently configured intermediate string for the b_defaultMessageHandler.
returns: The intermediate String.

b_setDefaultMessageHandlerPrefix [message type] [prefix] *blib/b_setDefaultMessageHandlerPrefix*
Set the prefix to use for the default message handler and the given message type.
[message type]: 0=informational message, 1=error message.
[prefix]: A string to prefix all messages of the given type.
returns: Nothing.

b_getDefaultMessageHandlerPrefix [message type] *blib/b_getDefaultMessageHandlerPrefix*
Get the prefix used for the default message handler and the given message type.
[message type]: 0=informational message, 1=error message.
returns: The currently configured prefix.

b_info [message] [first part] [last part] [message handler param 1] ... [message handler param n] *blib/b_info*
Send an informational message to the user. The message is dispatched via the currently configured message handler (default: b_defaultMessageHandler).
[message]: to send
[first part]: If set to 0, assume that the given message is the first part of an overall chain of messages (default: 0).
[last part]: If set to 0, assume that the given message is the final part of an overall chain of messages (default: 0).

[message handler param i]: Arbitrary parameters to pass to the currently configured message handler (cf. b_setMessageHandler).
returns: Nothing, always sets a zero exit code.

b_error [message] [first part] [last part] [message handler param 1] ... [message handler param n] *blib/b_error*
Send an error message to the user *without* erroring out. The message is dispatched via the currently configured message handler (default: b_defaultMessageHandler).
99.9% of all users will want to use the combination of B_ERR and B_E for proper error handling instead.
[message]: to send
[first part]: If set to 0, assume that the given message is the first part of an overall chain of messages (default: 0).
[last part]: If set to 0, assume that the given message is the final part of an overall chain of messages (default: 0).
[message handler param i]: Arbitrary parameters to pass to the currently configured message handler (cf. b_setMessageHandler).
returns: Nothing, always sets a zero exit code.

b_setMessageHandler [handler] *blib/b_setMessageHandler*
Set the handler used to send messages to the user. By default, b_defaultMessageHandler is used.
[handler]: Function to handle the messages. See b_defaultMessageHandler for the requirements.
returns: Nothing.

b_getMessageHandler *blib/b_getMessageHandler*
Get the currently configured message handler.
returns: The message handler.

b_enforceUser [user name] *blib/b_enforceUser*
enforce that the user is the given one and if not, exit the script and set a non-zero status code
[user name]: user name to check against
returns: nothing
@B_E

b_isFunction [potential function name] *blib/b_isFunction*
check whether the given function is defined
returns: zero exit code if the function is defined

b_getBlibModules *blib/b_getBlibModules*
get all available blib module names as a newline-separated list
returns: all available blib module names as newline-separated list

b_listContains [list] [entry] *blib/b_listContains*
check whether the given list contains the given entry
[list]: newline-separated list
[entry]: string to be found on a single line within the list (equality check)
returns: a zero exit code if the list contains the entry; a non-zero exit code otherwise

b_deps [dependency 1] ... [dependency n] *blib/b_deps*
Assert that the given dependencies are met and error out with B_E otherwise. This function is meant to be used by modules or scripts to declare all of their dependencies.
[dependency i]: Command that is absolutely required to run this script.
returns: Nothing. Errors out with B_E, if dependencies are not met.
@B_E

b_import [module] [double import] *blib/b_import*
Import the given module into the current context.
[module]: relative path of the module to import (relative to the blib/lib root directory)
[double import]: if set to 1, import the given module regardless of whether it was imported before (default: 0 = don't do duplicate imports)
returns: nothing, errors out if the import failed and sets a non-zero status code; if the import was successful or previously done, a zero exit code is set
@StateChanging
@B_E

b_generateStandalone [function] [module dep 1] .. [module dep n] - [function dep 1] .. [function dep d] - [function param 1] .. [function param p] *blib/b_generateStandalone*
Create a standalone variant of blib in a single file running the given function when called (sourcing that file will only make the functions available) and print that file to stdout.

The current execution state is not retained.
[function]: The function to call when the generated script is executed. All script parameters when calling [output file] are passed to this function. The function must be available in the current context.
[module dep i]: Names of the modules to include in the standalone file. They do not need to be imported. They are loaded in the specified order.
[-]: Dash used as separator between the various types of arguments. If none is provided, all parameters are assumed to be modules.
[function dep j]: An arbitrary number of functions that need to be added in order to satisfy the dependencies of the function to call (e.g. if function A is meant to be called, but uses function B internally, you'll have to pass B as one of its dependencies). Dependencies that can be found in added modules should *not* be added.

[function param p]: Static parameters to add to the function as single String. Dynamic parameters should be passed to the generated script.
returns: Sets a zero exit code and prints the output file to stdout on success. May error out otherwise.
@B_E

b_execFuncInCurrentContext [function] [module dep 1] .. [module dep n] - [function param 1] .. [function param p] *blib/b_execFuncInCurrentContext*
Execute the given function in the current context.
[function]: The function to execute.
[module dep i]: Names of the modules required by the function. They do not need to be imported by the function itself.
[-]: A dash as separator character between the various parameters.
[function param p]: An arbitrary number of function parameters.
returns: Whatever the executed function returns.

b_execFuncAs [user] [function] [module dep 1] .. [module dep n] - [function dep 1] .. [function dep d] - [function param 1] .. [function param p] *blib/b_execFuncAs*
Attempt to execute the Bash function as the given user.

Whether or not this works highly depends on the underlying OS and its (sudo & su) configuration. In particular this function may cause further execution to wait for the user to type in the password of the requested user.

If the given user is identical to the current user, b_execFuncAs may decide to run the function in the current context. Otherwise it may run in a different process, i.e. all initialization and state may be lost. Thus make sure to create any required state within your function.
[user]: User to execute the function as (default: root).
[function]: The function to execute.
[module dep i]: Names of the modules required by the function. They do not need to be imported by the function itself. They are loaded in the specified order.
[-]: A dash as separator character between the various parameters.
[function dep j]: An arbitrary number of functions that need to be added in order to satisfy the dependencies of the function to call (e.g. if function A is meant to be called, but uses function B internally, you'll have to pass B as one of its dependencies). Dependencies that can be found in added modules should *not* be added.
[function param p]: An arbitrary number of function parameters.
returns: Whatever the executed function returns. A non-zero exit code may also indicate that the user switch didn't work. In particular B_E is *not* called if the executed function returns an error.
@B_E

b_isModule [module name] *blib/b_isModule*
Test whether the given name represents a blib module name.
returns: sets a zero exit code if the given name is a valid module name

args

Stateful argument parser for bash.

Regular arguments and options are parsed via b_args_parse. Afterwards they can be retrieved via b_args_get and b_args_getOption. Options may have parameters, can be repeated and combined.

The module can also check the correctness of options. All remaining correctness checks (e.g. for regular or option parameters) are left to the user of this module.

Conventions:

- Options always start with `-` and may occur everywhere.
- Single character options may be combined, e.g. `-ajh` will be considered the same as `-a -j -h`. Long options such as `--option` cannot be combined. *Recommendation*: Use long options for those options which require parameters and single letter options for everything else.
- Everything after a space-separated double dash (`--`) is not considered an option, but a regular argument.

Copyright (C) 2020 David Hobach LGPLv3
0.9

Dependencies

no dependencies

Imports

types

Global Variables

B_ARGS *args/B_ARGS*
Array of regular/non-option arguments in the order of their appearance.
Instead of using it directly, it is recommended to use b_args_get instead. The array may be removed in future versions.

B_ARGS_OPTS *args/B_ARGS_OPTS*
Map of options. `[option]_[index]` is used as key with `[index]` starting at zero. The values are the option parameters (if any). Multiple option parameters are separated by tabs.
The latter separator can be changed via b_args_setOptionParamSeparator before calling b_args_parse.

The index will only increase if options are repeated.
Instead of using it directly, it is recommended to use b_args_getOption instead.
The array may be removed in future versions.

Functions

b_args_setOptionParamSeparator [separator] *args/b_args_setOptionParamSeparator*
Set the separator for multiple option parameters.
[separator]: String to use as separator for multiple option parameters.
returns: Nothing.

b_args_getOptionParamSeparator *args/b_args_getOptionParamSeparator*
Get the separator for multiple option parameters.
returns: Nothing.

b_args_init [allow flag] [option 1] [option param count 1] ... [option n] [option param count n] *args/b_args_init*
Initialize the args module. It is recommended to call this function before using this module, if you want to achieve any non-default behaviour.
[allow flag]: If set to 0, assume that non-specified options do not have any parameters (default). If set to 1, enforce that only the given options are allowed and otherwise error out.
[option i]: An allowed option including its leading - prefix.
[option param count i]: Number of expected parameters for that option. The parameters must directly follow the option. If less parameters are found, the parser will error out.
returns: Nothing, always sets a zero exit code.
@StateChanging
@B_E

b_args_parse [arguments] *args/b_args_parse*
Parse the given arguments.
Call b_args_init before parsing, if you desire any non-default parsing behaviour.
[arguments]: The arguments meant to be parsed, usually "$@".
returns: Sets a zero exit code on success and calls B_E otherwise.
@StateChanging
@B_E

b_args_assertOptions [option 1] ... [option n] *args/b_args_assertOptions*
Assert that the parsed options contain only the given allowed options (or less).
[option i]: Option to check against.
returns: Sets a zero exit code, if the current parsed state contains only allowed options and errors out with B_E otherwise. Prints a list of invalid options in the error case.
@B_E

b_args_get [index] [fallback] *args/b_args_get*
Get the argument at the given index.
[index]: Index of the argument to retrieve, starting at 0.
[fallback]: Value to return, if the given index was provided as empty argument or is missing. Default: empty String
returns: The argument at the given index. An *empty* (existing!) argument will cause the fallback to be returned with a zero exit code. If no argument was found at that index, a nonzero exit code is set and the fallback is returned.

b_args_getInt [index] [fallback] *args/b_args_getInt*
Convenience wrapper for b_args_get that also checks the type of the argument to be an integer.
returns: See b_args_get. Also sets a nonzero exit code, if the argument is not an integer.

b_args_getCount *args/b_args_getCount*
Get the number of arguments.
returns: Number of arguments.

b_args_getOption [option] [fallback] [repeat index] [parameter index] *args/b_args_getOption*
Check whether the given option is set and retrieve its parameter, if it was.
[option]: String defining the option, e.g. `--option` or `-a`.
[fallback]: Value to return if the option did not have any parameter or was not set. Default: empty String
[repeat index]: Index of the option to retrieve, if the option was repeated multiple times (default: 0 = first option).
[parameter index]: Index of the option parameter to retrieve, starting at 0 (default: return all option parameters, separated by b_args_getOptionParamSeparator).
returns: The option parameter, if the option was set. The fallback is returned and a zero exit code is set, if the option was set, but an *empty* (existing!) parameter was provided. A nonzero exit code indicates that the option was not set (the fallback is still returned).

b_args_getOptionInt [option] [fallback] [repeat index] [parameter index] *args/b_args_getOptionInt*
Convenience wrapper for b_args_getOption that also checks the type of the option to be an integer.
returns: See b_args_getOption. Also sets a nonzero exit code, if the option is not an integer.

b_args_getOptionCount *args/b_args_getOptionCount*
Get the number of options that were set.
returns: Number of options (incl. repeated options).

b_args_getAll [exclude regex 1] ... [exclude regex n] *args/b_args_getAll*
Retrieve all arguments (excl. options).
[exclude regex i]: Arguments matching any of the given regular expressions are excluded from the return value.
returns: All arguments as a single escaped string that can be used to pass them to functions or other scripts (without quotes).

b_args_getAllOptions [exclude regex 1] ... [exclude regex n] *args/b_args_getAllOptions*
Retrieve all options.
[exclude regex i]: Options matching any of the given regular expressions are excluded from the return value.
returns: All options as a single escaped string that can be used to pass them to functions or other scripts (without quotes).

arr

Collection of array related functions.

Copyright (C) 2018 David Hobach LGPLv3
0.2

Dependencies

no dependencies

Imports

no imports

Functions

b_arr_join [delimiter] [array] *arr/b_arr_join*
Join the given array; elements are separated with the given delimiter. The array is not checked to exist.
[delimiter]: String to use as delimiter.
[array]: Expanded array to join, e.g. "${arr[@]}".
returns: Joined version of the array. The exit code is always zero.

b_arr_toList [array] *arr/b_arr_toList*
Create a newline-separated list from the given array.
[array]: Expanded array to join, e.g. "${arr[@]}".
returns: List version of the array. The exit code is always zero.

b_arr_contains [element] [array] *arr/b_arr_contains*
Check whether an array contains an element.
[element]: element to check for its existence in the array

[array]: expanded array to check, e.g. "${arr[@]}"
returns: an exit code of 0, if the element was found and 1 otherwise

b_arr_mapsAreEqual [map spec 1] [map spec 2] *arr/b_arr_mapsAreEqual*
Check whether the two given maps/associative arrays are equal.
[map spec 1]: First map specification to check. Since maps cannot be passed directly to functions in Bash 4.2, you'll have to use `"$(declare -p "yourmap")"` instead.
[map spec 1]: Second map specification to check.
returns: an exit code of 0, if the maps are equal and 1 otherwise; B_E is only triggered on programming errors
@B_E

cdoc

Generate code documentation in many formats (e.g. html, pdf, manpage, ...) from code comments.

Lines applicable for the documentation in your code are assumed to match static (configurable) regular expressions. These lines are then fed to pandoc in order to generate a single html page (or pdf, manpage, ...) as documentation. If no conversion is required (input format = desired output format), pandoc is bypassed.

It should be possible to use this way of generating code documentation with most programming languages (incl. bash). The defaults however are set for bash and the blib way of documenting its code, i.e. you'll have to use the getters and setters of this module if you want something different. For instance the default is to check for lines starting with #+ (a special bash comment line) and add everything afterwards to the output documentation.

Various callback functions can be used to add content to the output of b_cdoc_generate. See the documentation of that function for details.

If you wish to create code documentation for your bash project in blib style, please use `../util/blib-cdoc`.

0.6

Dependencies

cat
mktemp
mv
rm

Imports

fs

Functions

b_cdoc_setExtractionRegex [regex] *cdoc/b_cdoc_setExtractionRegex*
Set the regular expression used to check for matching lines in code files. The first match (${BASH_REMATCH[1]}) is added to the documentation output.
returns: nothing
@StateChanging

b_cdoc_getExtractionRegex *cdoc/b_cdoc_getExtractionRegex*
See the setter.
returns: The property that was set.

b_cdoc_setFileCallback [callback function name] *cdoc/b_cdoc_setFileCallback*
Set the function to call by b_cdoc_generate exactly once before starting to process a source code file.

The callback function can be used to filter certain files from processing or add them as-is.
It should be declared as follows:

```
callback_function_name [file] [output format]
[file]:           The next file to process is passed here.
[output format]:  chosen output format
returns:          Nothing. Possible exit codes:
                  0 = continue normal processing (default)
                  1 = include the file as-is without any processing
                  2 = silently ignore that file / do not process it
                  other = abort all further processing with an error
```

returns: nothing
@StateChanging

b_cdoc_getFileCallback *cdoc/b_cdoc_getFileCallback*
See the setter.
returns: The property that was set.

b_cdoc_setDocumentBeginCallback [callback function name] *cdoc/b_cdoc_setDocumentBeginCallback*
Set the function to call by b_cdoc_generate exactly once right before it starts generating the output document.

The callback function should be declared as follows:

```
callback_function_name [document output file] [document output format]
[document output file]: path to the document output file
          (may not exist and should not be written to)
[document output format]: chosen output format
returns:  whatever should be added at the beginning of the output document;
          a non-zero exit code will abort further processing
```

returns: nothing
@StateChanging

b_cdoc_getDocumentBeginCallback *cdoc/b_cdoc_getDocumentBeginCallback*
See the setter.
returns: The property that was set.

b_cdoc_setPostProcessingCallback [callback function name]
cdoc/b_cdoc_setPostProcessingCallback
Set the function to call by b_cdoc_generate each time a code file was fully processed.

The callback function should be declared as follows:

```
callback_function_name [processed input] [input file] [document output format]
[processed input]: Everything that was found to match the
                   extraction regex in the [input file] by b_cdoc_generate.
[input file]:      The original input file.
[document output format]: chosen output format
returns:           whatever should be added to the output document for the
                   given input file (usually the processed input or some filtered
                   version of it); a non-zero exit code will abort further processing
```

returns: nothing
@StateChanging

b_cdoc_getPostProcessingCallback *cdoc/b_cdoc_getPostProcessingCallback*
See the setter.
returns: The property that was set.

b_cdoc_setDocumentEndCallback [callback function name]
cdoc/b_cdoc_setDocumentEndCallback
Set the function to call by b_cdoc_generate exactly once right after it generated the output document.

The callback function should be declared as follows:

```
callback_function_name [document output file] [document output format]
[document output file]: path to the document output file
                        (may not exist and should not be written to)
[document output format]: chosen output format
```

```
returns:                whatever should be added to the end of the output
                        document; a non-zero exit code will abort further processing
```

returns: nothing
@StateChanging

b_cdoc_getDocumentEndCallback *cdoc/b_cdoc_getDocumentEndCallback*
See the setter.
returns: The property that was set.

b_cdoc_setBlockCallback [callback function name] *cdoc/b_cdoc_setBlockCallback*
Set the function to call by b_cdoc_generate each time it hits a block of matching comments.

The callback function should be declared as follows:

```
callback_function_name [block] [input file] [document output format]
[block]:          The full block of documentation that was identified.
[block counter]: Number of blocks previously seen.
[input file]:     The original input file.
[document output format]: chosen output format
returns:          whatever should be added instead of the given block;
                  a non-zero exit code will abort further processing
```

returns: nothing
@StateChanging

b_cdoc_getBlockCallback *cdoc/b_cdoc_getBlockCallback*
See the setter.
returns: The property that was set.

b_cdoc_generate [input files] [output file] [output format] [additional pandoc options] *cdoc/b_cdoc_generate*
Generate a documentation file from the given list of input files or directories.

The concept is really simple: Each *block* of documentation will trigger the b_cdoc_getBlockCallback exactly once and you may add additional parsing logic on a block-wise level.
The following example will call the block callback twice (once with three `block 1` lines and once with two `block 2` lines):

```
#+block 1
#+block 1
#+block 1

#say hello
echo "hello world"
```

```
#+block 2
#+block 2
```

The other callback functions may be used for further processing.

[input files]: Newline-separated list of files or directories to generate the documentation from. The given order is respected; directories are recursively searched for files. It is currently assumed that these files are encoded in UTF-8.
[output file]: Path to the documentation file to generate. Should not exist.
[output format]: The target format of the documentation to generate. See pandoc for a list of available output formats. If none is specified, pandoc is bypassed and the input format is chosen as output format. Passing "pandoc" will let pandoc decide based on the extension of the output file.
[additional pandoc options]: All remainining parameters will be directly passed to pandoc. If none are provided, -s is implicitly added as default.
returns: Sets a non-zero exit code and exits the script on errors. Output from pandoc and other calls may be printed. Otherwise nothing is returned.
@B_E

b_cdoc_generateBlibStyle [input files] [output file] [output format] [delete existing] *cdoc/b_cdoc_generateBlibStyle*
A convenience wrapper for b_cdoc_generate which sets various reasonable parameters depending on the output format.
[input files]: see b_cdoc_generate
[output file]: where to write the generated output documentation to
[output format]: currently one of raw|html|pdf|man is supported (default: raw)
[delete existing]: whether or not to delete previously created output files (default: true/0); if set to false (1), the function will error out if a previously created file was found
returns: full path to the created documentation file on success; otherwise the function may error out
@B_E

Callback Functions b_cdoc_cbPrintNewline

cdoc/b_cdoc_cbPrintNewline
Prints a newline character.
returns: nothing

b_cdoc_cbPrintFirstParam [param]

cdoc/b_cdoc_cbPrintFirstParam
Prints the first parameter.
[param]: The parameter to print.
returns: the first parameter

daemon

Module providing access to a single background process providing some service (daemon). Exiting the foreground control process will *not* terminate the background process. Attempting to start multiple background daemons will be prevented in a thread-safe way.

Each background process is assumed to implement a `daemon_main` function and must be identified by a unique String.

If you need to exchange data with the background service, please have a look at the multithreading/ipcv or multithreading/ipcm modules.

Known Issue:
Since the daemon cannot detach from its session via `setsid` in bash, the daemon will remain in the process group of its parent. So killing the parent with e.g. Ctrl-C will cause the parent group including the daemon to terminate. As a workaround, users may either ignore SIGINT requests or make sure the parent exits as soon as possible (recommended).

0.5

Dependencies

kill
mkdir
mktemp
rm
umask

Imports

fd
multithreading/mtx
proc

Global Variables

B_DAEMON_ID *daemon/B_DAEMON_ID*
Contains the ID of the daemon, if and only if the current process is the daemon process.

Functions

b_daemon_init [quiet flag] [main name] [stdout file] [stderr file] [umask setting] *daemon/b_daemon_init*
Init the module paramters. It is necessary to call this method *before* using any other of this module unless you want to use the default paramters.

[quiet flag]: If set to 0 (default), don't print anything to stdout during the execution of start|stop|restart|status. Otherwise use b_info to print informational messages.
[main name]: Name of the main loop function to execute in the background process. Returning from that function will exit the background process. Default: `daemon_main`
[stdout file]: Where the background process should write its stdout stream to (default: /dev/null). Lines are appended.
[stderr file]: Where the background process should write its stderr stream to (default: /dev/null). Lines are appended.
[umask settings]: The umask settings to apply to the daemon (default: 0).
returns: Nothing.
@StateChanging

b_daemon_start [id] [arg 1] ... [arg n] *daemon/b_daemon_start*
Start the background process.
If you need to start it as a different user, simply run this function as a different user with e.g. (b_execFuncAs)[#b_execFuncAs]. Please keep in mind that control processes must have the permission to send signals to the daemon PID though.
[id]: Unique identifying String of the daemon to distinguish it from others.
[arg i]: An arbitrary number of arguments which can be passed to the main loop.
returns: Sets a zero exit code on success. Otherwise sets a non-zero exit code. In particular B_E is called, if the daemon is already running.
@B_E

b_daemon_stop [id] [termination signal] [kill timeout] *daemon/b_daemon_stop*
Stop the background process.
[id]: Unique identifying String of the daemon to distinguish it from others.
[termination signal]: A number or string specifying the signal to send to the daemon (default: 15 / SIGTERM). See `kill -l` for an overview.
[kill timeout]: Time in seconds after which the background process will be killed, if it remains unresponsive to the termination signal (default: 0 = wait indefinitely).
returns: Sets a zero exit code, if the daemon terminated by itself. An exit code of 2 indicates that the daemon had to be killed. An exit code of 3 means that it wasn't running. B_E is called on unexpected errors.
@B_E

b_daemon_restart [id] [termination signal] [kill timeout] [arg 1] ... [arg n] *daemon/b_daemon_restart*
Restart the background process.
[id]: Unique identifying String of the daemon to distinguish it from others.
[termination signal]: See b_daemon_stop.

[kill timeout]: See b_daemon_stop.
[arg i]: An arbitrary number of arguments which can be passed to the main loop.
returns: See b_daemon_start.
@B_E

b_daemon_statusPid [id] *daemon/b_daemon_statusPid*
Check the status of the background process.
Doesn't print informational messages to stdout.
[id]: Unique identifying String of the daemon to distinguish it from others.
returns: The PID and sets a zero exit code, if the daemon is running and a non-zero exit code otherwise. B_E is only called on exceptional errors.
@B_E

b_daemon_status [id] *daemon/b_daemon_status*
Check the status of the background process and print informational messages to stdout (if configured).
[id]: Unique identifying String of the daemon to distinguish it from others.
returns: Sets a zero exit code, if the daemon is running and a non-zero exit code otherwise. B_E is only called on exceptional errors.
@B_E

b_daemon_getPid [id] *daemon/b_daemon_getPid*
Get the process ID of the background process.
[id]: Unique identifying String of the daemon to distinguish it from others.
returns: The process ID and sets a zero exit code, if it could be obtained. Please note that the process may be dead anyway (use b_daemon_statusPid for that). Otherwise a non-zero exit code is set. B_E is only called on exceptional errors.
@B_E

b_daemon_sendSignal [id] [signal] *daemon/b_daemon_sendSignal*
Send a signal to the daemon.
For termination signals, please use b_daemon_stop instead.
[id]: Unique identifying String of the daemon to distinguish it from others.
[signal]: A number or string specifying the signal to send to the daemon. See `kill -l` for an overview.
returns: Sets a zero exit code, if the daemon was running and a non-zero exit code otherwise. B_E is only called on exceptional errors.
@B_E

date

Collection of date and time related functions.

0.3

Dependencies

date

Imports

no imports

Functions

b_date_add date [time] [unit] [format] [utc flag] *date/b_date_add*
Add the given number of seconds to the given date.
[date]: date to add seconds to; the format must be understood by the Unix date utility
[time]: amount of time to add
[unit]: unit of the time to add, may be one of d (days), h (hours) m (minutes), s (seconds, default)
[format]: output format of the date, in Unix date notation (default: use the localized output)
[utc flag]: if set to 0, use UTC as time zone if not specified for the input *and* use it for the output (default: 1 = local time zone)
returns: The input date with the given number of seconds added, in the requested format; returns a non-zero exit code on errors.
@B_E

b_date_addDays date [days] [format] [utc flag] *date/b_date_addDays*
Convenience wrapper to b_date_add with days.
[date]: See b_date_add.
[days]: Number of days to add.
[format]: See b_date_add.
[utc flag]: See b_date_add.
returns: See b_date_add.
@B_E

b_date_diff [date 1] [date 2] [unit] *date/b_date_diff*
Get the amount of time between the two dates, i.e. [date 2] - [date 1].
[date 2], [date 1]: the two dates to subtract; the time part is assumed to be identical if not specified within the dates
[unit]: unit of the result, may be one of d (days), h (hours) m (minutes), s (seconds, default)
returns: The amount of time between the given two dates [date 2] - [date 1], rounded down. Returns a non-zero exit code on errors.
@B_E

b_date_getFileModAge [file] [unit] *date/b_date_getFileModAge*
Get the time that passed since the last modification of the file.

[file]: Full path to the file to check.
[unit]: unit of the time to retrieve, may be one of d (days), h (hours) m (minutes), s (seconds, default)
returns: The amount of time in the given unit since the last modification. May be rounded down. Sets a non-zero exit code on errors.
@B_E

delay

Simplistic module to delay commands to a future time.
Requires polling.

All timestamps in this module must be integers and a larger timestamp must denote a time *after* a smaller timestamp.

0.3

Dependencies

no dependencies

Imports

no imports

Global Variables

B_DELAY_EXECUTED *delay/B_DELAY_EXECUTED*
The number of commands executed during the last invocation of b_delay_execute.

Functions

b_delay_to [timestamp] [command] *delay/b_delay_to*
Delay the given command to be executed at the given time.
[timestamp]: An integer timestamp (e.g. $SECONDS, Unix timestamp in s/ms/ns, ...).
[command]: The command to execute at the given time.
returns: Nothing.
@StateChanging

b_delay_execute [timestamp] *delay/b_delay_execute*
Execute all commands which are due at the given time.
[timestamp]: Integer timestamp representing the current point in time.
returns: Nothing. The exit code is equal to the number of commands with a non-zero exit code. B_DELAY_EXECUTED is updated with the number of

commands executed.
@StateChanging

b_delay_getCommandAt [timestamp] *delay/b_delay_getCommandAt*
Get the set of commands to be executed at the given point in time.
[timestamp]: Integer timestamp denoting the time for which to retrieve the commands.
returns: The commands to execute at that time.

dmcrypt

Abstraction layer for cryptsetup / dm-crypt.

Features:
- automatic management of dm-crypt devices
- password support for non-tty environments

Copyright (C) 2020 David Hobach LGPLv3
0.6

Dependencies

dirname
head
mkdir
readlink

Imports

hash
ui

Functions

b_dmcrypt_init [ui mode] *dmcrypt/b_dmcrypt_init*
Initialize this module. This function *must* be called at least once before using any of the other functions.
[ui mode]: How to request a password from the user: auto|gui|tty (default: auto).
returns: Nothing.
@StateChanging
@B_E

b_dmcrypt_getMapperName [path] *dmcrypt/b_dmcrypt_getMapperName*
Get the name of the dm-crypt mapper for a given path.
[path]: Full path to an encrypted file.
returns: A mapper name. This doesn't necessarily mean that the encrypted container is open. Use b_dmcrypt_isOpen for that.
@B_E

b_dmcrypt_createLuks [path] [size] [fs type] [entropy source] [password prompt] [dm-crypt option 1] ... [dm-crypt option n] *dmcrypt/b_dmcrypt_createLuks*
Create an encrypted luks container file at the given location.
This function may request a password from the user and usually requires root access rights.
[path]: Full path to the encrypted file to create.
[size]: Filesystem size in bytes to create. Supported suffixes: b 512, kB 1000, K 1024, MB 1000*1000, M 1024*1024, GB 1000*1000*1000, G 1024*1024*1024
[fs type]: Filesystem to create inside the encrypted container. If none is specified (default), no file system is created.
[entropy source]: Source of entropy to use for the file setup (default: /dev/urandom).
[password prompt]: Prompt string to ask the user for his password (optional).
[dm-crypt option i]: These options are directly passed to `cryptsetup`.
returns: Sets a zero exit code on success and errors out with B_E otherwise.
@B_E

b_dmcrypt_open [path] [mount point] [output var] [password prompt] [dm-crypt option 1] ... [dm-crypt option n] *dmcrypt/b_dmcrypt_open*
Open/Decrypt the given container and optionally mount it.
[path]: Full path to the encrypted file.
[mount point]: Where to mount the decrypted data (optional). If no mount point is specified, it will not be mounted.
[output var]: The name of the variable to write the created device to (optional).
[password prompt]: Prompt string to ask the user for his password (optional).
[dm-crypt option i]: These options are directly passed to `cryptsetup`.
returns: Sets a zero exit code on success.
@B_E

b_dmcrypt_close [path] [dm-crypt option 1] ... [dm-crypt option n] *dmcrypt/b_dmcrypt_close*
Close the given encrypted container.
[path]: Full path to the encrypted file.
[dm-crypt option i]: These options are directly passed to `cryptsetup`.
returns: A zero exit code on success. The exit code may also be zero for non-existing or already closed containers.
@B_E

b_dmcrypt_isOpen [path] *dmcrypt/b_dmcrypt_isOpen*
Check whether the given encrypted container is open (not necessarily mounted).
[path]: Full path to the encrypted file.
returns: Sets a zero exit code, if and only if the container is open. B_E is only called for exceptional errors.
@B_E

fd

Collection of file descriptor related functions.

0.3

Dependencies

no dependencies

Imports

no imports

Functions

b__fd__getOpen [pid] *fd/b__fd__getOpen*
Retrieve all open file descriptors for the given PID.
[pid]: process ID (default: `$$`)
returns: Newline-separated list of open file descriptor numbers; a non-zero exit code indicates that the process could not be found.

b__fd__closeNonStandard *fd/b__fd__closeNonStandard*
Close all non-standard file descriptors (i.e. those > 2) held by the current process.
returns: Nothing. B__E is called on unexpected errors.
@B__E

b__fd__closeAll *fd/b__fd__closeAll*
Close all file descriptors held by the current process.
returns: Nothing. B__E is called on unexpected errors.
@B__E

flog

Flexible log writer for bash.

Features:
- arbitrary output support (files, network streams, stdout, stderr, ...) in a user-defined format
- optional log file reduction
- optional thread safety
- support for partial messages

In order to log to the system log, please use the logger command instead. This library is mostly meant for application logs handled in a more custom manner.

Exact format of log entries:

```
[header][message]
[header]: Can be arbitrarily defined in the
          respective callback function. If nothing
          is defined, the below default header
          is used:
[default header] = '[default date] '
[default date]: current date in the format as used
          by date +"%F %T %Z" (the format can be changed)
```


0.6

Dependencies

cat
date
mkdir
mktemp
readlink
rm
tail

Imports

fs
hash
multithreading/mtx

Global Variables

B_FLOG_SEV *flog/B_FLOG_SEV*
Global map for human readable severities which may be used by users of this script.
It was inspired by the severities of RFC5424.
Currently supported values: emergency|alert|critical|crit|error|err|warning|warn|notice|informational|info|debug

Functions

b_flog_printSeverity [severity] *flog/b_flog_printSeverity*
[severity]: see b_flog_init
Print the given severity in a way for logging. This function is meant to be used as building block for header functions.
returns: a printed version of the given severity for logging

b_flog_close *flog/b_flog_close*
close the currently open log; is automatically called, but users may want to call it themselves to force the respective file descriptor to be closed before the program

is ended
returns: nothing
@StateChanging

b_flog_init [log file name] [header callback function] [log reduction lines] [thread safe] [intermediate] *flog/b_flog_init*
Initialize this log writer. This function **must** be called before any others.
[log file name]: name of the log file to write to; special files such as /dev/stdout, /dev/stderr (default), /dev/tcp, /dev/udp are supported if your bash version supports them; the file doesn't need to exist
[header callback function]: optional name of the function to be called whenever a new log entry is generated; the function must be defined as follows:

```
[header callback function] [severity]
[severity]: see [b_flog_log](#b_flog_log)
returns: the full header meant to be used for the current moment in time
         with the given severity (without knowing the message details)
         and sets a non-zero exit code on errors; errors may cause the
         message to be logged without header
```

[log reduction lines]: if set to a positive integer, reduce the log file approximately to that number of lines during logging (default: 3000) - see b_flog_setLogReductionLinesApprox for details; this option has no effect on non-file outputs (stdout, network output, ...)
[thread safe]: Whether calls to b_flog_log should be thread safe (0) or not (default: 1 = not thread safe).
[intermediate]: String to use as intermediate separator when chaining partial log messages (default: b_getDefaultMessageHandlerIntermediate).
returns: sets a non-zero exit code on errors and may exit the script
@StateChanging
@B_E

b_flog_log [message] [severity] [first part] [last part] *flog/b_flog_log*
Log the given message with the given optional severity.
If the [thread safe] variant was chosen, may wait for other log sources to write their message first.
[message]: message to log
[severity]: users may pass arbitrary numbers or even Strings here, but it is recommended to stick to the priorities defined in $BLIB_FLOG_SEV (default: ${B_FLOG_SEV["info"]})
[first part]: If set to 0, assume that the given message is the first part of an overall chain of messages (default: 0).
[last part]: If set to 0, assume that the given message is the final part of an overall chain of messages (default: 0). Please note that other threads may be blocked from writing to the output, if the last message of a chain was not yet received.

returns: Sets a non-zero exit code on errors. B_E is only called, if logging failed entirely.
@B_E

b_flog_messageHandler [message type] [message] [first part] [last part] *flog/b_flog_messageHandler*
A message handler implementation that handles messages by logging them via b_flog_log.
Issues with the logging system itself (e.g. log file not writable) are written to stderr.
If you don't want to log all messages and/or handle some of them differently, you can simply write a wrapper for this function.
[message type]: See b_defaultMessageHandler.
[message]: See b_defaultMessageHandler.
[first part]: See b_defaultMessageHandler.
[last part]: See b_defaultMessageHandler.
returns: Nothing. Never causes errors.

b_flog_getDateFormat *flog/b_flog_getDateFormat*
Get the date format used for the header by this log writer (see "man date" for explanations).
returns: see above

b_flog_setDateFormat [format string] *flog/b_flog_setDateFormat*
Set the date format used for the header by this log writer (see "man date" for explanations).
returns: nothing
@StateChanging

b_flog_getLogReductionLinesLowerBound *flog/b_flog_getLogReductionLinesLowerBound*
Get the number of lines that the log file will at least have after a log file reduction.
returns: see above

b_flog_getLogReductionLinesUpperBound *flog/b_flog_getLogReductionLinesUpperBound*
Get the maximum number of lines that the log file will have before it is reduced.
returns: see above

b_flog_setLogReductionLinesLowerBound [bound] *flog/b_flog_setLogReductionLinesLowerBound*
Set the number of lines that the log file will at least have after a log file reduction.
[bound]: number of lines to use for that bound
returns: nothing
@StateChanging

b_flog_setLogReductionLinesUpperBound *flog/b_flog_setLogReductionLinesUpperBound*
Set the maximum number of lines that the log file will have before it is reduced.
[bound]: number of lines to use for that bound
returns: nothing
@StateChanging

b_flog_setLogReductionLinesApprox [line count] *flog/b_flog_setLogReductionLinesApprox*
Set the number of average number of lines that the log file should have; counts <= 0 indicate no limit.
[line count]: reduce the log after reaching 1.2*[line count] lines to 0.8*[line count] lines
returns: nothing
@StateChanging

b_flog_getHeaderFunction *flog/b_flog_getHeaderFunction*
Get the name of the header callback function that is used.
returns: see above

b_flog_setHeaderFunction [header function] *flog/b_flog_setHeaderFunction*
Set the name of the header callback function to be used.
[header function]: name of the header function to use
returns: nothing
@StateChanging

Header Functions b_flog_defaultHeader [severity]

flog/b_flog_defaultHeader
Default header callback function used with b_flog_init.
[severity]: the default header ignores the severity
returns: the default header meant to be used for the current moment in time

b_flog_headerDateSeverity [severity]

flog/b_flog_headerDateSeverity
An alternative to the default header callback function which appends the severity to the default header.
[severity]: see b_flog_init
returns: the default header with the severity appended

b_flog_headerDateScriptSeverity [severity]

flog/b_flog_headerDateScriptSeverity
An alternative to the default header callback function which appends the calling script and the severity to the default header.
[severity]: see b_flog_init
returns: the default header with the calling script and severity appended

fs

Collection of file and file system related functions.

0.4

Dependencies

cat
findmnt
head
lsblk
mktemp
mount
rm
stat
sync
wc

Imports

no imports

Functions

b_fs_isEmptyDir [dir] *fs/b_fs_isEmptyDir*
Check whether the given directory is empty or non-existing. It is not checked whether the passed parameter is a file preventing a directory from being created.
[dir]: full path to the directory to check
returns: a zero exit code if the directory does not exist or is empty

b_fs_getLineCount [file] *fs/b_fs_getLineCount*
Get the number of lines of the given file.
[file]: full path to a file
returns: the number of lines; a non-zero exit code is set on errors
@B_E

b_fs_waitForFile [file] [maximum time] *fs/b_fs_waitForFile*
Sleep until the given file appears. The check interval is 1s.
[file]: full path to the file or directory to wait for
[maximum time]: maximum time in s to wait for the file to appear (default: forever)
returns: Sets a zero exit code if the file appeared and a non-zero exit code on a timeout.

b_fs_getMountpoints [device] *fs/b_fs_getMountpoints*
Get all mountpoints for the given device.
[device]: Full path to the device (incl. /dev/) for which to obtain the mountpoints.
returns: A newline-separated list of mountpoints where the given device is mounted to. Sets a non-zero exit code if no such mountpoints were found.

b_fs_mountIfNecessary [device] [mount point] [enforce] [mount options] *fs/b_fs_mountIfNecessary*
Mount the given device if it isn't already mounted.
[device]: Full path to the device (incl. /dev/) to mount.
[mount point]: Full path where to mount the device. If no mount point is specified, a /tmp/ mount point is chosen. Non-existing directories are created. Is ignored if another mount point already exists.
[enforce]: If set to 0, enforce the given mount point to be used in addition to potentially existing ones (default: 1). Mount options are ignored.
[mount options]: Options to pass to `mount`, if it needs to be executed (default: none).
returns: The chosen mount point or a newline-separated list of existing mount points on success; sets a non-zero exit code on failure.
@B_E

b_fs_createLoopDeviceIfNecessary [file] *fs/b_fs_createLoopDeviceIfNecessary*
Create a loop device for the given file if no old one exists. Usually requires root access rights.
[file]: File for which to create a loop device.
returns: Created loop device or previously used one (incl. /dev/). Sets a non-zero exit code, if no device could be created.
@B_E

b_fs_removeUnusedLoopDevice [device|file] *fs/b_fs_removeUnusedLoopDevice*
Remove a loop device, if and only if it is unused by the operating system. Otherwise mark it for removal once it becomes unused. Usually requires root access rights.
[device|file]: Full path (incl. /dev/) to the loop device or to the backing file.
returns: Sets a zero exit code, if the loop device was not used and could thus be successfully removed *or* does not exist. A nonzero exit code indicates that the device is still being used. B_E is called on other errors.
@B_E

b_fs_parseSize [string] [check flag] *fs/b_fs_parseSize*
Parse human-readable file system sizes that include units.
[string]: A string denoting a file system size of the format [number][unit]. Unit may be one of KB 1000, K 1024, MB 1000*1000, M 1024*1024, GB 1000*1000*1000, G 1024*1024*1024, and so on for T, P. If no unit is provided, the number is assumed to denote bytes. The number must be an integer.

[check flag]: Check whether the result makes sense (default: 0/true). This check will make integer overflows less likely.
returns: The respective number of bytes meant with the given string. B_E is called on parsing errors.
@B_E

b_fs_removeWithOverwrite [file] [randomness source] *fs/b_fs_removeWithOverwrite*
Overwrite the given file with random data, then remove it. This is meant to prevent a potential reconstruction of the file after its removal.
Warning: The reconstruction may still work on some types of file systems or physical storage systems (e.g. on flash disks/SSDs).
[file]: Full path to the file to remove. Directories are currently not supported.
[randomness source]: Device to use as source of random data (default: /dev/urandom).
returns: Nothing. B_E is called on errors.
@B_E

b_fs_isRotatingDrive [block device] *fs/b_fs_isRotatingDrive*
Check whether the given block device is a rotating disk (non-SSD) or not.
[block device]: Device name or path to check.
returns: Nothing. Sets a zero exit code, if the device is a rotating one. Non-existing devices trigger B_E.
@B_E

b_fs_getBlockDevice [file] *fs/b_fs_getBlockDevice*
Get the block device on which the given file or directory is stored.
[file]: Full path to the file.
returns: Full device path where the given file or directory is stored. May return pseudo file systems (e.g. tmpfs).
@B_E

b_fs_removeRelativelySafely [file] [randomness source] *fs/b_fs_removeRelativelySafely*
A best-effort implementation that attempts to remove files from the file system in a non-reconstructible way.
Contrary to b_fs_removeWithOverwrite it'll attempt to achieve similar results on SSDs.
May require root privileges.
Warning: The reconstruction may still work depending on your hardware.
[file]: Full path to the file to remove. Directories are currently not supported.
[randomness source]: Device to use as source of random data (default: /dev/urandom).
returns: Nothing. B_E is called on errors.
@B_E

b_fs_enumerate [path list] [nonexisting] *fs/b_fs_enumerate*
Enumerate all files found in the given list of paths. Recurse into directories as necessary.
[path list]: Newline-separated list of files and directories.
[nonexisting]: 0 = Error out with B_E on non-existing files (default). 1 = Silently drop non-existing files. 2 = Include non-exiting files in the output.
returns: List of files found in all of the given paths.
@B_E

hash

Hash functions.

Copyright (C) 2020 David Hobach LGPLv3
0.5

Dependencies

no dependencies

Imports

no imports

b_hash_file [file] [algorithm]

hash/b_hash_file
Compute the hash of the given file.
[file]: Full path to the file.
[algorithm]: Algorithm to use. Currently supported: md5|sha1|sha224|sha256|sha384|sha512|crc|blake2 (default: md5).
returns: The hash of the given file.
@B_E

b_hash_str [string] [algorithm]

hash/b_hash_str
Compute the hash of the given string.
[string]: String which to compute the hash for.
[algorithm]: Algorithm to use. Currently supported: md5|sha1|sha224|sha256|sha384|sha512|crc|blake2 (default: md5).
returns: The hash of the given string.
@B_E

http

Collection of http related functions.

0.2

Dependencies

curl

Imports

types

Global Variables

B_HTTP_CHECKURLS *http/B_HTTP_CHECKURLS*
Each call to b_http_getOnlineStatus causes one of the URLs in this array to be visited.
It is recommended to pick a relatively large number of URLs with SSL support to remain relatively anonymous, if b_http_getOnlineStatus is called multiple times.
Defaults to the European/US Alexa Top 10 (which hopefully blends in to the masses).

Functions

b_http_rawUrlEncode [string] *http/b_http_rawUrlEncode*
Encode the given string according to RFC 3986.
[string]: to encode
returns: Returns a string in which all non-alphanumeric characters except -_.~ have been replaced with a percent (%) sign followed by two hex digits. This is the encoding described in RFC 3986 for protecting literal characters from being interpreted as special URL delimiters, and for protecting URLs from being mangled by transmission media with character conversions (like some email systems). A non-zero exit code is set on errors.
@B_E

b_http_rawUrlDecode [string] *http/b_http_rawUrlDecode*
Decode the given string encoded with b_str_rawUrlEncode or an equivalent function.
[string]: to decode
returns: The literal string with all hex characters replaced; a non-zero exit code is set on errors.

b_http_getOnlineStatus [timeout] *http/b_http_getOnlineStatus*
Find out whether we are online or not by attempting an http connection.
One of B_HTTP_CHECKURLS is possibly visited during the process.
Use b_net_getDNSStatus for DNS-only checks.
[timeout]: Timeout in seconds for hanging checks (default: 5).

returns: 0, if we're online, 1 if only DNS works, 2 if neither DNS nor http(s) worked, 3 if the check timed out; B_E will be called if the status cannot be determined.
@B_E

b_http_testProxy [proxy string] [intransparent only] [timeout]
http/b_http_testProxy
Test whether the given proxy is working as advertised.
[proxy string]: Connection string to test, e.g. `https://1.1.1.1:234`.
[intransparent only]: If set to 0 (default), only accept intransparent proxies (i.e. those not revealing your IP).
[timeout]: Time in seconds after which to consider the proxy non-responsive (default: 5).
returns: Nothing, but sets a zero exit code, if and only if the proxy appears to work.

ini

Stateful ini reader for bash.

Currently only a single file per instance of this library/thread is kept in memory, but you can read multiple files one after another or in multiple threads.

Implementation Specifics:

- names/keys & values are case sensitive
- comment lines may start with ; or #
- whitespace lines are ignored
- duplicate names may result in undefined behaviour (usually the second will override the first)
- all characters following the = are considered part of the value (incl. whitespace); whitespace before and after the value may be trimmed by the getters though (check their description)
- values are not interpreted (e.g. quotes, escape characters, ...)
- whitespace around keys and around section qualifiers is ignored

0.5

Dependencies

no dependencies

Imports

no imports

Functions

b_ini_read [ini file] *ini/b_ini_read*
read the given ini file and keep it in thread-local memory so that subsequent calls to the b_ini_get functions will return the values from the ini file; subsequent calls to this function will update the internal state to represent the file last read in this thread
[ini file]: path to the ini file to read
returns: an error message on errors and sets a non-zero exit code on errors
@StateChanging
@B_E

b_ini_get [name] [section] *ini/b_ini_get*
get the value for the ini entry with the given name as String in raw format
[name]: name/key of the ini entry to retrieve
[section]: section where to look for the entry with the given name (default: without section)
returns: value of the ini entry matching exactly the given section and name incl. any whitespace; a non-zero exit code is set if such an entry wasn't found

b_ini_getString [name] [section] *ini/b_ini_getString*
get the value for the ini entry with the given name as String and remove all whitespace around the returned String
[name]: name/key of the ini entry to retrieve
[section]: section where to look for the entry with the given name (default: without section)
returns: value of the ini entry matching exactly the given section and name excl. any whitespace around; a non-zero exit code is set if such an entry wasn't found

b_ini_getInt [name] [section] *ini/b_ini_getInt*
get the value for the ini entry with the given name as integer
[name]: see b_ini_get
[section]: see b_ini_get
returns: see b_ini_get; additionally it is checked whether the return value is an integer (if not, a non-zero exit code of 2 is set and the return value is undefined)

b_ini_getBool [name] [section] *ini/b_ini_getBool*
get the value for the ini entry with the given name as boolean
[name]: see b_ini_get
[section]: see b_ini_get
returns: see b_ini_get; 0 is returned via echo for true, 1 for false; the exit code indicates a potential error during parsing (2) or a missing entry (1) and *not* true/false

b_ini_assertNames [section 1] [name 1] ... [name n] – [section 2] [name 1] ... [name m] *ini/b_ini_assertNames*
Assert that the given sections contains *at most* the given names and no additional ones.
This is a useful function to detect user mistakes and should be called right after b_ini_read.
Multiple sections can be separated with "–".
[section]: see b_ini_get
[name]: see b_ini_get
returns: Sets a zero exit code if and only if the last read ini file doesn't contain any additional names. Otherwise B_E is triggered. B_ERR will list the invalid names.
@B_E

keys

Simple *system-wide* cryptographic key store protected by a master password.

A key store is only opened when necessary. It'll remain open from then on until an application calls b_keys_close.

The default key store can be found at `/etc/blib/keys`. Applications may deploy their own exclusive key store, if necessary (not recommended).

Usually requires root access rights.

Features:
- thread safe: no write operation can happen to a closed key store (b_keys_close and write operations are single threaded only)
- retrieved keys are read-only unless managed via this interface

Copyright (C) 2022 David Hobach LGPLv3
0.7

Dependencies

cat
cp
find
findmnt
mkdir
readlink
rm

Imports

dmcrypt
multithreading/mtx

Functions

b_keys_getDefaultStore *keys/b_keys_getDefaultStore*
Get the system-wide default key store directory.
returns: The default store directory. Always sets a zero exit code.

b_keys_init [app id] [auto create] [ui mode] [password prompt] [wait time] [store dir] *keys/b_keys_init*
Initialize this module. This function *must* be called at least once before any other function except b_keys_create can be used.
Will open the key store, if it's closed.
[app id]: Unique application ID (recommended: $B_SCRIPT_NAME).
[auto create]: Automatically create a new key store, if none exists (default: 0 = do it). Users may create their own key store using `cryptsetup` otherwise.
[ui mode]: How to request the master password from the user: auto|gui|tty (default: auto).
[password prompt]: Prompt string to ask the user for his password (optional).
[wait time]: Maximum time in ms to wait for another process writing to a key (default: 300) (-1 = indefinitely).
[store dir]: Full path to a directory where to manage the keys (default: b_keys_getDefaultStore). Only applications requiring an exclusive key store should use a non-default value here.
returns: Sets a zero exit code on success and errors out with B_E otherwise.
@StateChanging
@B_E

b_keys_add [key id] [key path] *keys/b_keys_add*
Add the given key to the key store.
[key id]: Unique identifier for the key to add.
[key path]: Full path to the key to add. It is *not* removed. Use functions such as b_fs_removeRelativelySafely to do that.
returns: Sets a zero exit code on success and errors out with B_E otherwise.
@B_E

b_keys_get [key id] *keys/b_keys_get*
Retrieve the given key from the store. If you need the content as String, please use b_keys_getContent.
[key id]: Unique identifier for the key to retrieve.
returns: Path to the key. It may not exist, if the key store was closed in the meantime or the provided ID does not exist.
@B_E

b_keys_getAll [global] *keys/b_keys_getAll*
Retrieve all keys from the store.
[global]: If set to 0, also retrieve keys for other application IDs (default: 1).

returns: All key paths for reading as a newline-separated list. If the key store is closed or modified during the runtime, the list may be empty or incomplete.
@B_E

b_keys_getContent [key id] *keys/b_keys_getContent*
Retrieve the given key content from the store. For binary keys please use b_keys_get.
[key id]: Unique identifier for the key to retrieve.
returns: The key content as String. If it doesn't exist or other errors occurred, B_E is called.
@B_E

b_keys_delete [key id] [backup] *keys/b_keys_delete*
Delete the given key from the key store.
[key id]: Unique identifier for the key to delete.
[backup]: Whether (0) or not (1) to create a backup (default: 0).
returns: Sets a zero exit code on success and errors out with B_E otherwise.
@B_E

b_keys_close [store dir] *keys/b_keys_close*
Close the key store, if necessary. Works without initialisation (which might open the key store), if necessary.
WARNING: 99% of all applications should *never* call this function as it may block all subsequent read operations by any other applications using this system-wide key store. The only applications that might call this function are key store management applications and only upon explicit user request. The key store is otherwise automatically closed upon system shutdown by the running OS.
[store dir]: The main directory of the store to close (default: the key store directory used in b_keys_init or - if not initialised - b_keys_getDefaultStore).
returns: Sets a zero exit code on success (closed key store) and errors out with B_E otherwise.
@B_E

meta

Functions providing information about blib modules or other bash scripts.

0.2

Dependencies

no dependencies

Imports

str

Functions

b_meta_getClearImports [path] *meta/b_meta_getClearImports*
Get the list of modules that the given script imports via b_import.
Important: All b_import calls are assumed to be declared on a dedicated line and to start at the first character of that line ("clear" import). This is considered a feature to evade this method (for doc purposes mostly).
[path]: Full path to the script to retrieve the module imports for.
returns: Newline-separated list of imports of the given script.
@B_E

b_meta_getClearDeps [path] *meta/b_meta_getClearDeps*
Get the list of dependencies that the given script declares via b_deps.
Important: All b_deps calls are assumed to be declared on a dedicated line and to start at the first character of that line ("clear" dependency). This is considered a feature to evade this method (for doc purposes mostly).
[path]: Full path to the script to retrieve the dependencies for.
returns: Newline-separated list of dependencies of the given script.
@B_E

multithreading/ipcm

An inter-process map implementation.

The map is available to all running processes during a single boot session.

Both reading and writing to the map can be done from any number of processes.

Overall Features	
# readers	multiple
# writers	multiple
read consistency	always
write consistency	always
blocking	on writes

0.2

Dependencies

no dependencies

Imports

multithreading/ipcv
multithreading/mtx

Functions

b_ipcm_setNamespace [namespace] *multithreading/ipcm/b_ipcm_setNamespace*
Set the common process namespace to use. All processes inside the same namespace share a common state.
It is recommended to call this function a single time before any other functions of this module. Otherwise the default namespace, which may include unrelated processes, is used.
[namespace]: Name of the namespace to set.
returns: Errors out, if the name is unacceptable.
@StateChanging
@B_E

b_ipcm_getNamespace *multithreading/ipcm/b_ipcm_getNamespace*
Retrieve the currently used namespace.
returns: The currently used namespace.

b_ipcm_change [key] [change function] [maximum time] *multithreading/ipcm/b_ipcm_change*
Change the given key/value combination inside the map in a thread-safe way. May wait for changes done by other processes.
[key]: A global unique identifier for the given value.
[change function]: Name of the function to execute the change of the value. It will be called with the current key as first parameter and the current value as second. It is expected to print the new value to set for this key. A non-zero exit code will cause b_ipcm_change to abort the change.
[maximum time]: maximum time in ms to wait for other processes to complete their operation (default: -1 = indefinitely)
returns: A zero exit code and prints the new value, if the change succeeded. An exit code of B_RC+1 indicates that the change function returned a non-zero exit code. B_E is called otherwise.
@B_E

b_ipcm_get [key] [fallback] *multithreading/ipcm/b_ipcm_get*
Retrieve the data found at the given key.
[key]: A global unique identifier for the data to retrieve.
[fallback]: Data to return if nothing was found for the given key (default: empty).
returns: Sets a zero exit code and returns the data found on success. If no data was found, the fallback data is returned and a zero exit code is set. B_E is called on errors.

b_ipcm_unsetNamespace [namespace] [maximum time] *multithreading/ipcm/b_ipcm_unsetNamespace*
Unsets the given namespace and all keys stored within it.
It is recommended to call this function when all processes finished their work.

[namespace]: The namespace to unset (default: the current namespace).
[maximum time]: maximum time in ms to wait for other processes to complete their operation (default: 0 = indefinitely)
returns: Sets a zero exit code only upon successful removal. Otherwise B_E is triggered.
@B_E

multithreading/ipcv

Provides means for inter-process communication (ipc) via global bash variables (v).

This implementation uses shared memory, i.e. it should be reasonably fast.

Only a single process or thread is assumed to be writing (i.e. use b_ipcv_save) a variable at a time and multiple processes may read it (using e.g. b_ipcv_load). If you need to write a single variable from multiple processes, please consider using the multithreading/mtx module or similar locking means in combination with this module (or just multithreading/ipcm).

Overall Features	
# readers	multiple
# writers	single
read consistency	always
write consistency	only for one writer
blocking	never

0.3

Dependencies

findmnt
mkdir
mktemp
mv
rm

Imports

no imports

Functions

b_ipcv_save [namespace] [var name 1] .. [var name n] *multithreading/ipcv/b_ipcv_save*

Save the current values of the given variables so that they are made available for other processes under the given namespace.

Please note that each variable is saved atomically, but individually. I.e. if you need multiple values to be updated at the same time, please use a single variable (e.g. a map).
[namespace]: Name for a common group under which the given variables should be saved. The combination of [namespace] and [variable name] must be a unique identifier across all processes running on the system.
[var name i]: The name of the global variable to make accessible for other processes. An arbitrary number of variable names can be specified.
returns: Nothing, but a non-zero exit code indicates failed variable save attempts. A failed save attempt also triggers B_E.
@B_E

b_ipcv_load [namespace] [var name 1] .. [var name n] *multithreading/ipcv/b_ipcv_load*
Load the given variables from the given namespace into the current process context.
[namespace]: Name of the group under which the variable was saved with b_ipcv_save. Must exist.
[var name i]: Name of the variable to load. Multiple names can be specified.
returns: A non-zero exit code indicates the number of variables that could not be loaded unless some unexpected error occurred and B_E is triggered. Please note that a failed load attempt/unavailable variable does generally not trigger B_E.
@B_E

b_ipcv_loadNamespace [namespace] [check existence] *multithreading/ipcv/b_ipcv_loadNamespace*
Load all variables that can be loaded for the given namespace into the current process context.
[namespace]: Name of the group for which to load all available variables.
[check existence]: Whether or not to make sure that the namespace to load exists (default: 0/check). Otherwise non-existing namespaces will not cause an error.
returns: Sets a zero exit code on success. Failing to load any single available variable will always trigger B_E.
@B_E

b_ipcv_unset [namespace] [var name 1] .. [var name n] *multithreading/ipcv/b_ipcv_unset*
Unset/Remove the given variables from the global namespace.
Please note that the variables will remain set in your current process context, if they were set before. Use the standard bash `unset` for that.
[namespace]: Group where the given variables belong to.
[var name i]: Name of the variable to remove. Multiple may be specified.

returns: The number of variables which could not be unset. B_E is not triggered for these.
@B_E

b_ipcv_unsetNamespace [namespace] *multithreading/ipcv/b_ipcv_unsetNamespace*
Remove the given global namespace and all variables it contains.
Please note that the variables will remain set in your current process context, if they were set before. Use the standard bash `unset` for that.
[namespace]: To remove.
returns: Sets a zero exit code only upon successful removal. Otherwise B_E is triggered.
@B_E

multithreading/mtx

Collection of mutex related functions.

Mutex: Only a single process may have it at any point in time.
Semaphore: A specific maximum number of processes may have it at any point in time.

See the keys module source code for an example on how to use this mutex implementation efficiently.

0.3

Dependencies

cat
mkdir
mktemp
rm
rmdir
sleep
touch

Imports

proc

Functions

b_mtx_setSleepTime [ms] *multithreading/mtx/b_mtx_setSleepTime*
Sets the time to sleep for this module whenever active polling is done (default: 500).
[ms]: time in miliseconds between active polling requests for e.g. mutexes done

by this module; must be an integer
returns: Nothing, always sets a zero exit code.

b_mtx_getSleepTime *multithreading/mtx/b_mtx_getSleepTime*
Gets the time to sleep for this module whenever active polling is done.
returns: The currently set time to sleep in ms.

b_mtx_create [base dir] *multithreading/mtx/b_mtx_create*
Allocate a new mutex without claiming it (use b_mtx_try for that).
[base dir]: Path to an *existing* directory where to store the mutex (default: not specified). By default this module will pick a temporary location. If you need a mutex that persists across reboots, please set a directory that persists across reboots here. The path should point to a local, non-network file system destination. The module must be able to create remove files or directories there at will.
returns: A string identifying the mutex (mutex ID). Sets a non-zero exit code on errors.
@B_E

b_mtx_release [mutex] [block ID] *multithreading/mtx/b_mtx_release*
Release the given mutex so that it can be used by other block IDs/threads.
[mutex]: A mutex obtained via b_mtx_create.
[block ID]: The block ID for which to release the mutex (default: $$).
returns: Sets a non-zero exit code if the mutex could not be removed as another process is blocking it and a zero exit code on successful removal.

b_mtx_forceRelease [mutex] *multithreading/mtx/b_mtx_forceRelease*
Release the given mutex so that it can be used by other blockIDs/threads.
Warning: This function can remove mutexes from other threads and should generally *only* be used for the removal of mutexes which are known to be stale by the calling application.
[mutex]: A mutex obtained via b_mtx_create.
returns: Nothing and sets a zero exit code.

b_mtx_pass [mutex] [block ID] *multithreading/mtx/b_mtx_pass*
Pass a blocked mutex to another block ID (i.e. change the block ID of the given mutex).
You should only do this if you currently own the mutex and the new process is ready to take over.
[mutex]: A mutex obtained via b_mtx_create.
[block ID]: The block ID to set for the given mutex.
returns: Sets a zero exit code on success and a non-zero exit code otherwise.
@B_E

b_mtx_try [mutex] [block ID] [claim stale] [claim own] *multithreading/mtx/b_mtx_try*
Attempt to obtain the given mutex. Return immediately even if it cannot be obtained.
[mutex]: A mutex obtained via b_mtx_create. You may also use a static and otherwise unused directory path as mutex and share it across all relevant processes.
[block ID]: The ID to use by which to block (default: running (sub)shell process id $$). This should be the process ID of the process attempting to obtain the mutex or you should know what you're doing. If you're in a subshell that should deploy a mutex against other subshells, store their $BASHPID and call the function with that.
[claim stale]: If set to 0, claim the mutex even if it is still blocked by some other process, but that process isn't running anymore. If set to 1 (default), the function returns without obtaining the mutex. In general this should only be used in situations where a mutex has a high probability of being stale (e.g. application start).
[claim own]: If set to 0 (default), claim the mutex if it appears to be blocked by the provided block ID. If set to 1, consider it blocked even then.
returns: The function incl. parameters to execute to remove the mutex if it was obtained and an error message stating the reason otherwise. The provided function *should* be called as part of an exit trap of the calling script or via eval. Sets an exit code of 0, if the mutex was obtained. An exit code of 1 is set, if the mutex was blocked and another non-zero exit code if some other error occurred (the mutex might be blocked even then).

Example code:

```
local mutex=""
local mutexRet=""
mutex="$(b_mtx_create)" || { B_ERR="Failed to create a mutex." ; B_E ; }
mutexRet="$(b_mtx_try "$mutex")" \
|| { B_ERR="Failed to obtain the mutex $mutex. Reason: $mutexRet" ; B_E ; }
#assuming the mutex is only meant to be removed after full
#execution of the script:
trap "$mutexRet" EXIT
#direct removal:
#b_mtx_release "$mutex"
```

b_mtx_waitFor [mutex] [block ID] [claim stale] [maximum time] *multithreading/mtx/b_mtx_waitFor*
Wait for the given mutex to become available. This will block script execution.
[mutex]: see b_mtx_try
[block ID]: see b_mtx_try
[claim stale]: see b_mtx_try
[maximum time]: maximum time in ms to wait for the mutex to become available

(default: -1 = indefinitely)
returns: see b_mtx_try

multithreading/multiw

Allow multiple processes to write to a *virtual* file at the same time without causing write inconsitencies (written data from each process mangled with each other).

This is achieved by keeping one file per process and relies on the assumption that both replacing and reading a symlink on your Linux distribution is atomic.

In order for this to work, all write operations must go through this module.

Currently reading only returns the data written by the process which wrote last. If you need some sort of appending, it makes more sense to deploy a mutex using e.g. the multithreading/mtx module.

Overall Features	
# readers	multiple
# writers	multiple
read consistency	partial, last writer wins
write consistency	always
blocking	never

0.2

Dependencies

ln
mv
rm
shuf
stat

Imports

no imports

Functions

b_multiw_setMaxHangTime [seconds] *multithreading/multiw/b_multiw_setMaxHangTime*
Set the maximum time that a process is expected to hang between two instructions. This is relevant for various internal guarantees.
[seconds]: Time in seconds that a process hangs at most.

returns: Nothing.
@StateChanging

b_multiw_getMaxHangTime *multithreading/multiw/b_multiw_getMaxHangTime*
Get the maximum time that a process is expected to hang between two instructions. This is relevant for various internal guarantees.
returns: Time in seconds.

b_multiw_write [file path] *multithreading/multiw/b_multiw_write*
Write all data lying in stdin to the given *virtual* file in a thread-safe way.
[file path]: Full path to the virtual file to write to. Must not be a regular file (but may not exist).
returns: A zero exit code, if the write operation was successful and a non-zero exit code otherwise.
@B_E

b_multiw_remove [file path] *multithreading/multiw/b_multiw_remove*
Remove the given virtual file *and all of its revisions*.
This function should only be called when all processes finished reading and writing. It is recommended to use it over the standard Linux `rm` as the latter will leave remnants behind.
returns: A zero exit code on success.
@B_E

net

Collection of network related functions.

0.1

Dependencies

dig

Imports

no imports

Global Variables

B_NET_CHECKHOSTS *net/B_NET_CHECKHOSTS*
Each call to b_net_getDNSStatus causes one of the host records in this array to be requested.
It is recommended to pick a relatively large number of URLs with SSL support to remain relatively anonymous, if b_net_getDNSStatus is called multiple times.

Defaults to the European/US Alexa Top 10 (which hopefully blends in to the masses).

B_NET_DNSSERVERS *net/B_NET_DNSSERVERS*
Each call to b_net_getDNSStatus with a `random` [server] causes one of the IPs in this array to be used as DNS server.
Defaults to common large-scale free DNS providers (that probably save your requests).
Sources:
- https://www.techradar.com/news/best-dns-server
- https://dnsmap.io/articles/most-popular-dns-servers

Functions

b_net_getDNSStatus [timeout] [server] *net/b_net_getDNSStatus*
Find out whether DNS appears to work or not by testing it.
One of B_NET_CHECKHOSTS is possibly requested during the process.
Use b_http_getOnlineStatus for http checks.
[timeout]: Timeout in seconds for hanging checks (default: 5).
[server]: DNS server to use (default: the OS default). Passing `random` will pick a random DNS server.
returns: 0 if DNS works as expected, 1 if DNS works but returns NXDOMAIN and 2 on timeout; B_E will be called if the status cannot be determined.
@B_E

notify

Collection of notification related functions.

Copyright (C) 2021 David Hobach LGPLv3
0.6

Dependencies

getent
id
notify-send
su

Imports

fs

Functions

b_notify_waitForUserDbus [user] [maximum time] *notify/b_notify_waitForUserDbus*
Wait for *some* dbus user session to come up at `/run/user/[uid]/bus`. As

dbus is required for notifications to work, waiting may be required for early notifications.
[maximum time]: Maximum time in s to wait for the file to appear (default: forever).
[user]: Name of the user whose instance to wait for (default: first user with a readable home directory).
returns: Sets a zero exit code and returns the identified dbus instance, if a dbus instance came up and a non-zero exit code on timeout. B_E is used for exceptional errors.
@B_E

b_notify_send [arg 1] ... [arg n] *notify/b_notify_send*
Send out a notification to the user via `notify-send`.
If run as root, may send the notification to *all* active non-root users.
Calling `notify-send` directly can be problematic depending on the user, dbus session, environment variables, ... This function aims to circumvent these potential issues.
[arg i]: All arguments are directly passed to `notify-send`.
returns: Nothing. Errors during notification sending will trigger B_E.
@B_E

b_notify_sendNoError [arg 1] ... [arg n] *notify/b_notify_sendNoError*
Convenience variant of b_notify_send that does not exit and only prints errors with b_defaultErrorHandler. This function should e.g. be used in custom error handlers to avoid recursion.
returns: Sets a non-zero exit code on errors.

os/osid

Functions for operating system identification.

Copyright (C) 2018 David Hobach LGPLv3
0.2

Dependencies

no dependencies

Imports

no imports

Functions

b_osid_init [force] *os/osid/b_osid_init*
[force]: if set to 0, force an init even if it would otherwise not be necessary (default: 1 - only initialize if it didn't happen before)

Initialize the osid module. It should normally *not* be necessary to call this function directly, but it will be called by the osid module internally as needed.
returns: May error out and set a non-zero exit code on failures.

b_osid_isDebian *os/osid/b_osid_isDebian*
Check whether the OS running this function is a Debian Linux.
returns: Sets a zero exit code if the check returns true. Does not print any output.

b_osid_isDebianLike *os/osid/b_osid_isDebianLike*
Check whether the OS running this function is a Debian Linux or one of its derivatives (e.g. ubuntu).
returns: Sets a zero exit code if the check returns true. Does not print any output.

b_osid_isOpenSuse *os/osid/b_osid_isOpenSuse*
Check whether the OS running this function is a OpenSUSE.
returns: Sets a zero exit code if the check returns true. Does not print any output.

b_osid_isFedora *os/osid/b_osid_isFedora*
Check whether the OS running this function is a Fedora Linux.
returns: Sets a zero exit code if the check returns true. Does not print any output.

b_osid_isCentOS *os/osid/b_osid_isCentOS*
Check whether the OS running this function is a CentOS.
returns: Sets a zero exit code if the check returns true. Does not print any output.

b_osid_isRedHat *os/osid/b_osid_isRedHat*
Check whether the OS running this function is a RedHat Linux.
returns: Sets a zero exit code if the check returns true. Does not print any output.

b_osid_isUbuntu *os/osid/b_osid_isUbuntu*
Check whether the OS running this function is an Ubuntu Linux.
returns: Sets a zero exit code if the check returns true. Does not print any output.

b_osid_isFedoraLike *os/osid/b_osid_isFedoraLike*
Check whether the OS running this function is a Fedora Linux or one of its derivatives (e.g. CentOS, Red Hat, Qubes OS).
returns: Sets a zero exit code if the check returns true. Does not print any output.

b_osid_isQubesDom0 *os/osid/b_osid_isQubesDom0*
Check whether the OS running this function is a Qubes OS in dom0.
returns: Sets a zero exit code if the check returns true. Does not print any output.

b_osid_isQubesVM *os/osid/b_osid_isQubesVM*
Check whether the OS running this function is a Qubes OS in a VM.
returns: Sets a zero exit code if the check returns true. Does not print any output.

os/qubes4/dom0

Collection of functions supporting scripting in Qubes OS 4.x dom0.

Important: Whenever you parse output from VMs to dom0, you **must** be extra careful and assume it totally untrusted as parsing bugs are a plausible attack vector for compromised VMs. Passing data to potentially compromised VMs of course also exposes that data's confidentiality.

0.7

Dependencies

basename
cat
dd
dirname
kill
losetup
mktemp
ps
python3
qubesdb-rm
qubesdb-write
qubes-prefs
qvm-block
qvm-check
qvm-firewall
qvm-ls
qvm-pool
qvm-run
qvm-shutdown
qvm-volume
readlink
setsid
sleep

sort
tar
tee
timeout
xxd

Imports

fs
proc
types
ui

Global Variables

B_DOM0_VM_STDERR *os/qubes4/dom0/B_DOM0_VM_STDERR*
Where to append the VM stderr output for b_dom0_qvmRun and its related functions.
Defaults to `/dev/null` for security reasons (VM output is always untrusted).

B_DOM0_QVM_RUN_PARAMS *os/qubes4/dom0/B_DOM0_QVM_RUN_PARAMS*
Alternative method to pass parameters to b_dom0_qvmRun and its related functions.
Useful for e.g. `--no-gui` to speed up performance when you never use a VM's GUI.
Default: Empty array.

Functions

b_dom0_setVMDeps [list] [keep defaults] *os/qubes4/dom0/b_dom0_setVMDeps*
Set the dependencies for all VMs used with this module to the given list of binaries.
These dependencies are checked before *any* execution of commands in *any* VM; in particular whenever b_dom0_qvmRun, b_dom0_execIn, b_dom0_execStrIn or b_dom0_execFuncIn are called.
[list]: Newline-separated list of binaries or commands that VMs must be able to execute.
[keep defaults]: If set to 0 (default), merge the default list of dependencies with the supplied one. Otherwise only keep the supplied list of dependencies.
returns: Nothing.
@StateChanging

b_dom0_getVMDeps *os/qubes4/dom0/b_dom0_getVMDeps*
Getter for b_dom0_setVMDeps.
returns: The current list of dependencies for VMs.

b_dom0_qvmRun [parameter 1] ... [parameter n] [vm] [command]
os/qubes4/dom0/b_dom0_qvmRun
A wrapper for qvm-run which sets reasonable defaults for shell scripting and applies various fixes.

Most calls to qvm-run should be made via this function rather than interacting with qvm-run directly as the Qubes OS qvm-run was designed with interactive shell usage in mind whereas this wrapper is intended for bash developers.

Particular features:

- a certain set of reasonable default parameters is used: -p -q -n -u root
- -n was set as auto-starting VMs during Bash scripting can heavily influence the user experience (imagine the VM being shut down manually by the user whilst a bash script is running –> constant restarts)
- stdin is redirected to /dev/null by default to avoid potential security implications (accidental reads from dom0 stdin passed to a VM); this can be overridden using –stdin
- stdout has the VM output and the exit code is the one of the VM
- stderr VM output is appended to B_DOM0_VM_STDERR (defult: `/dev/null` for security reasons)
- distinguished exit conditions (executed command failed vs. qvm-run failed)
- workarounds for known Qubes bugs wrt qvm-run may be implemented here (e.g. qubes issues #3083, #4476, #4633 in the past)
- VM dependencies set with b_dom0_setVMDeps are checked

Please note that calling this function will make your script wait for the execution of the commands in the client VM.

Wherever possible, this function should be combined with b_silence as the VM output shouldn't be trusted. Otherwise please keep in mind that **both** stdout *and* stderr may have untrusted output which may even contain binary data. In order to validate against binary data you can e.g. use b_types_parseString or b_types_str.

[parameters]: Any parameters supported by qvm-run. If you pass -a, the default -n will be overridden. If you pass -u, the default root user is overridden. If you pass -v, -q will be overridden. If you pass –stdin, even stdin is passed to qvm-run. -p can be overridden by using /dev/null redirection. Only the short parameter versions and only one switch per argument are supported. Parameters may also be globally specified via B_DOM0_QVM_RUN_PARAMS, but the global parameters are ignored, if any local parameters are specified.
[vm]: The VM where to run the given [command].
[command]: The command to run.
returns: Sets the exit code of qvm-run and prints its output. May error out using B_E if qvm-run itself fails.
@B_E

b_dom0_getDispVMs *os/qubes4/dom0/b_dom0_getDispVMs*
Get a list of all currently existing disposable VMs.
returns: The currently existing disposable VMs as newline-separated list.
@B_E

b_dom0_startDispVM [template] *os/qubes4/dom0/b_dom0_startDispVM*
Start a dispVM from the given template in the background and return its name. The disposable VM will remain started until it is shut down. If you only wish to execute a single command, please use b_dom0_qvmRun with the –dispVM parameter.
It may take a while for this function to obtain the name of the dispVM.
[template]: The template to use for the dispVM. If no template is specified, use the default Qubes template.
returns: Name of the dispVM that was started and sets a zero exit code on success. This function may error out.
@B_E

b_dom0_execIn [vm] [file] [user] *os/qubes4/dom0/b_dom0_execIn*
Execute the file as bash code in the given VM and wait for it to finish.

See b_dom0_qvmRun for various notes and words of caution.
[vm]: Name of the VM where to execute the given string. The VM is assumed to be started.
[file]: Bash file to execute in the given VM.
[user]: user as which to execute the bash file (default: root)
returns: Whatever the executed Bash code prints in the VM to stderr or stdout; the status code is set to the one of the executed Bash code on success (0). Non-zero exit codes and error messages may come from both this function as well as the code executed in the given VM.
@B_E

b_dom0_execStrIn [vm] [string] [user] *os/qubes4/dom0/b_dom0_execStrIn*
Execute the String as bash code in the given VM and wait for it to finish.

Convenience wrapper for b_dom0_execIn.

See b_dom0_qvmRun for various notes and words of caution.
[vm]: see b_dom0_execIn
[string]: Bash String to execute in the given VM.
[user]: see b_dom0_execIn
returns: see b_dom0_execIn; B_E is *not* called if the executed command returns an error
@B_E

b_dom0_execFuncIn [vm] [user] [function] [module dep 1] .. [module dep n] - [function dep 1] .. [function dep d] - [function param 1] ..

[function param p] *os/qubes4/dom0/b_dom0_execFuncIn*
Execute the Bash function in the given VM and wait for it to finish.

Convenience wrapper for b_dom0_execIn.

See b_dom0_qvmRun for various notes and words of caution.
[vm]: see b_dom0_execIn
[user]: see b_dom0_execIn
[function]: Name of the function as it is declared in the current scope.
[module dep i]: Names of the modules required by the function. They do not need to be imported by the function itself.
[-]: A dash as separator character between the various parameters.
[function dep j]: An arbitrary number of functions that need to be added in order to satisfy the dependencies of the function to call (e.g. if function A is meant to be called, but uses function B internally, you'll have to pass B as one of its dependencies). Dependencies that can be found in added modules must *not* be added.
[function param p]: An arbitrary number of function parameters.
returns: see b_dom0_execIn; B_E is *not* called if the executed command returns an error
@B_E

b_dom0_waitForFileIn [vm] [file] [maximum time] *os/qubes4/dom0/b_dom0_waitForFileIn*
Convenience wrapper for b_fs_waitForFile.
[vm]: VM where to execute.
[file]: See [b_fs_waitForFile].
[maximum time]: See [b_fs_waitForFile].
returns: See [b_fs_waitForFile].
@B_E

b_dom0_isMountedIn [vm] [device] *os/qubes4/dom0/b_dom0_isMountedIn*
Check whether the device is mounted in the given VM.
[vm]: VM where to execute.
[device]: Full path to the device (incl. /dev/) to check.
returns: Sets a zero exit code if the device is mounted in the VM; a non-zero exit code means that it's either not mounted or some other error occurred.
@B_E

b_dom0_mountIfNecessary [vm] [device] [mount point] [enforce] [mount options] *os/qubes4/dom0/b_dom0_mountIfNecessary*
Mount the given device in the target VM if it isn't already mounted there. Actually a wrapper for b_fs_mountIfNecessary.
[vm]: VM where to execute.
[device]: Full path to the device (incl. /dev/) to mount.
[mount point]: Full path where to mount the device. If no mount point is specified, a /tmp/ mount point is chosen. Non-existing directories are created.

Is ignored if another mount point already exists.
[enforce]: If set to 0, enforce the given mount point to be used in addition to potentially existing ones (default: 1).
[mount options]: Options to pass to `mount`, if it needs to be executed (default: none).
returns: The chosen mount point or a newline-separated list of existing mount points on success; sets a non-zero exit code on failure. As these strings are returned from the VM, extra care must be taken when parsing them.
@B_E

b_dom0_createLoopDeviceIfNecessary [vm] [file] *os/qubes4/dom0/b_dom0_createLoopDeviceIfNecessa*
Create a loop device for the file in the given VM if no old one exists. Actually a wrapper for b_fs_createLoopDeviceIfNecessary.
This usually requires root privileges.
[vm]: VM where to execute.
[file]: File for which to create a loop device.
returns: Created loop device or previously used one (incl. /dev/). Sets a non-zero exit code, if no device could be created.
@B_E

b_dom0_removeUnusedLoopDevice [vm] [device|file] [type] [map] *os/qubes4/dom0/b_dom0_removeUnusedLoopDevice*
Remove a loop device from the given VM, if it is *neither* used by Qubes OS *nor* by the given VM.
Important: Calling this function may cause the device to be removed as soon as it becomes unused by the VM (and the VM *only*!). So if you called this function, you should *not* use the device any further from dom0 with e.g. `qvm-block` afterwards - even if it wasn't removed.
[vm]: VM where to remove the loop device.
[device|file]: Full path (incl. /dev/) to the loop device or to the backing file inside the VM.
[type]: Whether the given path represents a loop device (0) or the backing file (1). If not specified, a device will be assumed, if and only if the path starts with `/dev/`.
[map]: Optional output from a previous call to b_dom0_parseQvmBlock. If none is specified, this function will internally call b_dom0_parseQvmBlock.
returns: Sets a zero exit code, if the loop device was not used and could thus be successfully removed *or* does not exist. A nonzero exit code indicates that the device is still being used. B_E is called on other errors.
@B_E

b_dom0_copy [dom0 file] [target VM] [target VM dir] [overwrite] [parent dir] *os/qubes4/dom0/b_dom0_copy*
Grab a file or directory in dom0 and push it to the given file path in the target VM.

[dom0 file]: location of the dom0 file or directory to read from, assumed to exist
[target VM]: VM to write to, assumed to exist. Must be started.
[target VM dir]: full path to the parent directory in the target VM to copy the file or directory to; non-existing parent directories are created; the name is taken from the name of the file/directory in dom0
[overwrite]: Whether or not to overwrite an existing [destination file] (default: 0 = overwrite).
[parent dir]: Set this to 0, if the [target VM dir] is the target *parent* directory (default) and to 1 if it includes the target file or folder *name* as last element.
returns: Sets an exit code of 0, if everything went fine, and a non-zero exit code otherwise.
@B_E

b_dom0_crossCopy [source VM] [source file] [target VM] [target VM dir] [overwrite] [parent dir] *os/qubes4/dom0/b_dom0_crossCopy*
Cross copy a file or directory from one VM to another, initiated by dom0. No user prompt is displayed.
[source VM]: Where to copy the source file from. Must be started.
[source file]: The file or directory to copy.
[target VM]: Where to copy the file to. Must be started.
[target VM dir]: full path to the parent directory in the target VM to copy the file or directory to; non-existing parent directories are created; the name is taken from the name of the file/directory in dom0
[overwrite]: Whether or not to overwrite an existing [destination file] (default: 0 = overwrite).
[parent dir]: Set this to 0, if the [target VM dir] is the target *parent* directory (default) and to 1 if it includes the target file or folder *name* as last element.
returns: Sets an exit code of 0, if everything went fine, and a non-zero exit code otherwise.
@B_E

b_dom0_ensureRunning [vm 1] ... [vm n] *os/qubes4/dom0/b_dom0_ensureRunning*
Start or unpause the given VMs if needed.
[vm i]: The VM to start.
returns: Nothing. B_E will be called for VMs that could not be started (maybe they don't exist?).
@B_E

b_dom0_isRunning [vm 1] ... [vm n] *os/qubes4/dom0/b_dom0_isRunning*
Check whether the given VMs are running *and* fully operational / not hanging / not paused / not booting.
In contrast `qvm-check --running [vm]` returns true for VMs which are currently booting and `qvm-ls` doesn't check whether the OS of a VM is hanging.
[vm i]: The VM to check.
returns: Nothing. B_E will be called for VMs that are not running (or don't

exist?).
@B_E

b_dom0_isHalted [vm 1] ... [vm n] *os/qubes4/dom0/b_dom0_isHalted*
Check whether the given VMs are halted.
This is not necessarily the inverse of b_dom0_isRunning as VMs may e.g. be paused.
[vm i]: The VM to check.
returns: Nothing. B_E will be called for VMs that are not halted (or don't exist?).
@B_E

b_dom0_ensureHalted [vm 1] ... [vm n] *os/qubes4/dom0/b_dom0_ensureHalted*
Shut down the given VMs if needed.
[vm i]: The VM to halt.
returns: Nothing. B_E will be called for VMs that could not be halted (maybe they don't exist?).
@B_E

b_dom0_exists [vm] *os/qubes4/dom0/b_dom0_exists*
Check whether the given VM exists.
[vm]: The VM to check.
returns: Sets a zero exit code, if the VM exists a non-zero exit code otherwise.

b_dom0_openCrypt [vm] [device] [mapper name] [rw flag] [mount point] [key file] [additional options] [password prompt] *os/qubes4/dom0/b_dom0_openCrypt*
In the given VM, open the given crypto device with dm-crypt and mount it to the mount point.
[vm]: The VM where to open the crypto device.
[device]: Full path to the device (incl. /dev/) to open.
[mapper name]: The name to assign to the decrypted version of the crypto block device. The created decrypted device will be found at */dev/mapper/[mapper name]*.
[rw flag]: 0=open read-write, 1=open read-only (default: 0)
[mount point]: Where to mount the decrypted data to. Non-existing directories will be created. If no mount point is specified, it will not be mounted (default).
[key file]: Full vm path to the key to use for decryption. If none is specified, password-based decryption is assumed and stdin will be read to obtain the password.
[additional options]: Single string with additional cryptsetup parameters to pass on (default: none). They are passed *as-is*, i.e. you'll have to take care of proper escaping etc. yourself.
[password prompt]: Optional string to use when the user is required to provide a decryption password.

returns: nothing (except for user interaction prompts if no key file is provided), but sets a non-zero exit code on errors
@B_E

b_dom0_closeCrypt [vm] [mapper name] [mount point] *os/qubes4/dom0/b_dom0_closeCrypt*
Close a crypto device opened with b_dom0_openCrypt.
[vm]: The VM where to close the crypto device.
[mapper name]: The name used when it was opened.
[mount point]: If the decrypted data is mounted inside the [vm], please specify the mount point here so that it can be unmounted before closing the device. Otherwise the function will attempt to close the device without unmounting (likely to fail).
returns: nothing, but sets a non-zero exit code on errors
@B_E

b_dom0_parseQvmBlock [variable name] [input] *os/qubes4/dom0/b_dom0_parseQvmBlock*
Parse data from `qvm-block ls` to an associative array.
The indices of the associative array will be of the format *[counter]_[field]* (counters run from 0 (inclusive) to max (exclusive)).
The special index "max" is equal to the number of lines. It can be used for iterations over the map.
Currently supported [field] values: backend|device id|id|description|used by|read-only|frontend-dev
[variable name]: Name of the associative array to use as output.
[input]: Optional output from a previous `qvm-block ls` call; if none is specified, this function will execute the call and use its output.
returns: A string specifying an associative array in bash syntax. You can eval that string to obtain all relevant data or use the more convenient b_dom0_getQvmBlockInfo. On errors B_E is called.
@B_E

b_dom0_getQvmBlockInfo [map] [retrieve field] [filter field 1] [filter value 1] .. [filter field n] .. [filter value n] *os/qubes4/dom0/b_dom0_getQvmBlockInfo*
Convenience function to retrieve information from the output of b_dom0_parseQvmBlock. Searches for the given filter values in the given fields and retrieves the first field value matching all filters.
More simple, but less flexible than b_dom0_parseQvmBlock.
[map]: Optional output from a previous call to b_dom0_parseQvmBlock. If none is specified, this function will internally call b_dom0_parseQvmBlock.
[retrieve field]: Name of the field to retrieve.
[filter field i]: Name of any field supported by b_dom0_parseQvmBlock.
[filter value i]: The value to search for in [filter field i] (equality check).
returns: The value of the requested field and sets a zero exit code on success. Sets a non-zero exit code if no matching value could be found. On errors B_E is called.

@B_E

b_dom0_attachDevice [dom0 device] [target VM] [rw flag] [force] *os/qubes4/dom0/b_dom0_attachDevice*
Attach the given device from dom0 (!) as block device to the target VM.
[dom0 device]: Full path to the device node (incl. /dev/) *in dom0*.
[target VM]: VM to attach the device to. Must be started.
[rw flag]: If set to 0, attaches the dom0 device in r/w (read-write) mode. If set to 1 (default), attaches the file in r/o (read only) mode.
[force]: If set to 0, attempt to force Qubes OS to recognize the device (default: 1). This is required for e.g. VM images. Please call b_dom0_detachDevice with the clean flag for the detach operation, if set to 0.
returns: The full path to the device created in the target VM and sets a zero exit code on success. Otherwise a non-zero exit code is set.
@B_E

b_dom0_attachFile [dom0 file] [target VM] [rw flag] [force] *os/qubes4/dom0/b_dom0_attachFile*
Attach the given file from dom0 (!) as block device to the target VM.
The function may attempt to acquire root privileges (and thus display a password prompt).
[dom0 file]: Full path to the file *in dom0* to attach.
[target VM]: VM to attach the file to. Must be started.
[rw flag]: If set to 0, attaches the dom0 file in r/w (read-write) mode. If set to 1 (default), attaches the file in r/o (read only) mode.
[force]: If set to 0, attempt to force Qubes OS to recognize the device (default: 1). This is required for e.g. VM images. Please call b_dom0_detachDevice with the clean flag for the detach operation, if set to 0.
returns: The full path to the device created in the target VM and sets a zero exit code on success. Otherwise a non-zero exit code is set.
@B_E

b_dom0_attachVMDisk [source VM] [target VM] [rw flag] [pool driver] [disk path] *os/qubes4/dom0/b_dom0_attachVMDisk*
Attach the entire private disk image (private.img) of the source VM to the target VM.
Warning: This is contradictory to all Qubes principles and should only be done if you know exactly what you're doing. Qubes OS even has some countermeasures to prevent accidental use of this feature which are bypassed here.
[source VM]: Name of the VM whose private disk to attach to the target VM. *All data* of that VM will be shared with the target VM. Will be shut down as part of this function and must remain shut down as long as the disk is attached.
[target VM]: VM where to attach the disk as block device to. Must be started.
[rw flag]: If set to 0, attaches the disk file in r/w (read-write) mode. If set to 1 (default), attaches the file in r/o (read only) mode.

[pool driver]: Driver of the pool that the disk was created in: One of file, file-reflink or lvm_thin (default: driver of the given [source VM]).
[disk path]: Full path to the disk. Default: Path to the private volume of the [source VM] in the pool of the respective [pool driver]. May override the [source VM] parameter.
returns: The full path to the device created in the target VM and sets a zero exit code on success. Otherwise a non-zero exit code is set. Please call b_dom0_detachDevice with the clean flag for the detach operation.
@B_E

b_dom0_getPoolDriver [vm] [volume] *os/qubes4/dom0/b_dom0_getPoolDriver*
Get the name of the pool driver used by the given VM.
[vm]: VM for which to retrieve the pool driver (default: system default pool).
[volume]: VM volume type for which to retrieve the pool driver (default: private).
returns: Name of the pool driver. B_E is called on errors.
@B_E

b_dom0_getDefaultPoolDriver [volume] *os/qubes4/dom0/b_dom0_getDefaultPoolDriver*
Retrieve the driver name of the system's default pool.
[volume]: Volume type for which to retrieve the pool driver (default: private).
returns: Driver name of the system's default pool.
@B_E

b_dom0_crossAttachDevice [source VM] [source device] [target VM] [rw flag] *os/qubes4/dom0/b_dom0_crossAttachDevice*
Attach the given block device from the source VM to the target VM.
This is merely a convenience wrapper for `qvm-block attach`.
[source VM]: VM where the source file can be found.
[source device]: Device to attach to the [target VM].
[target VM]: VM to attach the device to. Must be started.
[rw flag]: If set to 0, attaches the [source deivce] in r/w (read-write) mode. If set to 1 (default), attaches the file in r/o (read only) mode.
returns: The full path to the device created in the target VM and sets a zero exit code on success. Otherwise a non-zero exit code is set.
@B_E

b_dom0_crossAttachFile [source VM] [source file] [target VM] [rw flag] *os/qubes4/dom0/b_dom0_crossAttachFile*
Attach the given file from the source VM as block device to the target VM.
[source VM]: VM where the source file can be found.
[source file]: File to attach as block device.
[target VM]: VM to attach the file to. Must be started.
[rw flag]: If set to 0, attaches the [source file] in r/w (read-write) mode. If set to 1 (default), attaches the file in r/o (read only) mode.

returns: The full path to the device created in the target VM and sets a zero exit code on success. Otherwise a non-zero exit code is set.
@B_E

b_dom0_detachDevice [vm] [device] [clean flag] *os/qubes4/dom0/b_dom0_detachDevice*
Attempts to detach the given device from the VM. This may fail if the VM is using the device and thus it is usually a better idea to just shut the VM down.
[vm]: VM from which to detach the device.
[device]: Full path to the device in the VM. E.g. the return values of b_dom0_crossAttachFile, b_dom0_attachFile or b_dom0_attachVMDisk.
[clean flag]: Also wipe the device from the Qubes DB, if it's there (default: 1).
returns: nothing, but sets a zero exit code on success
@B_E

b_dom0_enterEventLoop [callback function] [heartbeat interval] *os/qubes4/dom0/b_dom0_enterEventLoop*
Enter a blocking loop to react to Qubes OS events. b_dom0_disconnectEventLoop can be used to disconnect from the event loop asynchronously, e.g. from exit traps.
If you want to obtain an overview of Qubes OS events, please use the `qwatch` utility manually.
[callback function]: Name of the function to call for *every* Qubes OS event. Since the number of events may be high, the function should do appropriate filtering at high performance.
The callback function should be declared as follows:

```
callback_function_name [subject] [event name] [event info] [timestamp]
[subject]: The subject name Qubes OS provides. Usually the VM for which the
    event was reported. 'None' appears to mean 'dom0'.
[event name]: Name of the event for which the callback function was called.
[event info]: May contain additional information about the event (e.g.
              arguments).
[timestamp]: When the event was received in ms since EPOCH.
returns:  Nothing. A non-zero exit code will abort further processing.
```

[heartbeat interval]: Interval in ms at which to send heartbeat events (default: no heartbeat events).
returns: Nothing, but uses the exit code of the last callback function execution as its own. Sets an exit code of 0, if the event loop was terminated by b_dom0_disonnectEventLoop. B_E is only called on exceptional errors.
@StateChanging
@B_E

b_dom0_disconnectEventLoop *os/qubes4/dom0/b_dom0_disconnectEventLoop*
Send a termination request to a dom0 event loop started with b_dom0_enterEventLoop. This is useful to exit the event loop from outside of it.

returns: A non-zero exit code indicates that either no active event loop existed or the signal sending failed. Never errors out to be suitable for e.g. exit traps.
@StateChanging

b_dom0_testMultiple [vm] [operator] [list] *os/qubes4/dom0/b_dom0_testMultiple*
Run multiple `test` checks inside the given VM with a single `qvm-run` call.
[vm]: Where to run the checks.
[operator]: The test operator to use for the list of parameters, e.g. `-f`.
[list]: Newline-separated list of parameters to check, e.g. a list of files for the `-f` operator.
returns: The subset of the list that matched the operator and sets a zero exit code on success.
@B_E

b_dom0_applyFirewall [vm] [rules] *os/qubes4/dom0/b_dom0_applyFirewall*
Apply the given rules to the VM firewall.
Important: Since the rules are applied iteratively, there may be short timeframes during which the VM may have no network access.
[vm]: VM for which to apply the firewall rules.
[rules]: Newline-separated list of firewall rules in *raw* format (cf. `man qvm-firewall`). The rules are added in the order as specified.
(This corresponds to the `qvm-firewall --raw [vm]` output.) Previously applied rules are left untouched.
Empty lines and lines starting with `#` (comments) are ignored.
returns: Nothing. If a rule cannot be added for whatever reason, a drop-all rule will be inserted in the beginning for security reasons.
@B_E

b_dom0_clearFirewall [vm] *os/qubes4/dom0/b_dom0_clearFirewall*
Clears the given VM firewall entirely (no rules left). This will make the VM firewall drop all connections.
Existing connections may continue to work as long as the VM and the connection remain up (this is a known Qubes OS quirk).
[vm]: VM for which to clear all rules.
returns: Nothing. Errors will result in a drop-all state.
@B_E

proc

Collection of process and thread related functions.

0.2

Dependencies

kill
killall
tail
timeout

Imports

no imports

Functions

b__proc__pidExists [pid] *proc/b_proc_pidExists*
Check whether the given process ID exists on the system.
[pid]: process ID to check for existence (process exists)
returns: A zero exit code, if it exists and a non-zero exit code if it doesn't; this function attempts to check the existence of the given process across *all* users, but it cannot guarantee correctness if the user running this script has very low privileges.

b__proc__childExists [pid] *proc/b_proc_childExists*
Check whether the given process ID exists *and* is a *direct* child of the calling bash process.
[pid]: process ID to check
returns: A zero exit code, if the caller is a parent of the given pid.

b__proc__waitForPid [pid] [maximum time] *proc/b_proc_waitForPid*
Wait for the given process to exit. If it doesn't exist, exit immediately.
[pid]: Process ID of the process to wait for.
[maximum time]: Maximum time in seconds to wait for the process to exit (default: 0 = indefinitely).
returns: Nothing, always sets a zero exit code. Use b_proc_pidExists if you need to know whether the process finished.

b__proc__resolveSignal [signal] *proc/b_proc_resolveSignal*
Resolve process (kill) signal names to their numeric identifiers.
[signal]: Signal name (string) or numeric identifier.
returns: The numeric identifier corresponding to the given signal.
@B_E

b__proc__killAndWait [pid] [signal] [timeout] *proc/b_proc_killAndWait*
Send a kill/exit signal to a process and wait for it to terminate.
[pid]: Process ID of the process to wait to terminate.
[signal]: Signal name (string) or numeric identifier to send (default: SIGTERM).
[timeout]: Time in seconds after which to send a SIGKILL if the process doesn't

terminate (default: 0 = wait forever).
returns: A zero exit code, if the process terminated without timeout or could not be found.
@B_E

b_proc_killByRegexAndWait [regex] [signal] [timeout] *proc/b_proc_killByRegexAndWait*
Kill/Terminate all matching processes.
[regex]: Regular expression matched against running process names. Matching processes will be terminated.
[signal]: Signal name (string) or numeric identifier to send (default: SIGTERM).
[timeout]: Time in seconds after which to send a SIGKILL if the processes don't terminate (default: 0 = wait forever).
returns: A zero exit code, if the process terminated without timeout or could not be found.
@B_E

sqlite3

Stateful sqlite driver for bash.

This module provides all features of the sqlite interactive mode to non-interactive bash scripts. See `man sqlite3` for the available commands.

Keeping a single database connection open usually exhibits better performance than calling `sqlite3` in batch mode over and over again.

Side Note: If you want to read or write csv files, this module can also do the job for you with standard SQL syntax. See `man sqlite3` on how to read and write csv files with sqlite.

Copyright (C) 2020 David Hobach LGPLv3
0.5

Dependencies

mkfifo
mktemp
rm
sqlite3

Imports

no imports

Functions

b_sqlite3_open [db file] [timeout] *sqlite3/b_sqlite3_open*
Open a new sqlite connection to a database. This **must** be called exactly once

before any calls to b_sqlite3_exec.
[db file]: The database file to connect to (default: a new in-memory database). If it doesn't exist, it may be created (the behaviour is identical to `sqlite3`).
[timeout]: Maximum time in seconds to wait for a command to execute on the database via b_sqlite3_exec. Default: -1 = indefinitely
returns: Nothing.
@StateChanging
@B_E

b_sqlite3_getOpen *sqlite3/b_sqlite3_getOpen*
Retrieve the currently open database.
returns: Prints the currently open database and an empty string for an unnamed database. Sets a zero exit code, if and only if a database is currently open.

b_sqlite3_exec [command] [timeout] [filter input] *sqlite3/b_sqlite3_exec*
Executes the given command on an open sqlite database (cf. b_sqlite3_open).
[command]: The command to execute. All commands supported by `sqlite3` in interactive mode incl. the SQL commands are supported. The only exception is the `.output` command. Please use the bash output redirection instead of `.output`. **Warning**: Incomplete (e.g. missing `;`) or incorrect commands may cause this function to make your bash script hang forever, if no [timeout] is specified.
[timeout]: Maximum time in seconds to wait for the command to execute on the database (default: the timeout initialized via b_sqlite3_open).
[filter input]: Whether or not to filter command input lines from the returned output (default: 0 = true).
returns: The output provided by the database in response to the executed command incl. potential errors. The function attempts to set a non-zero exit code, if the output contains error messages. B_E is only called on timeouts or other database connectivity issues.
@B_E

b_sqlite3_close *sqlite3/b_sqlite3_close*
Closes a currently open database connection.
It is highly recommended to execute this function once you don't need the database connection anymore (usually at the end of a script).
returns: Nothing. A non-zero exit code indicates a failed close operation.
@StateChanging

str

Collection of string related functions.

0.2

Dependencies

no dependencies

Imports

no imports

Functions

b__str__stripQuotes [string] *str/b__str__stripQuotes*
Remove any single or double quotes around the given string.
[string]: string which might be enclosed in single or double quotes (' or ")
returns: [string] without the enclosed single or double quotes, if there were any; if none were found the original string is returned; the exit code is always zero

b__str__trim [string] *str/b__str__trim*
Remove any whitespace from around a string.
[string]: string to trim
returns: [string] beginning and ending without whitespace; the exit code is always zero

b__str__prefixLines [string] [prefix] *str/b__str__prefixLines*
Prefix all lines of the given string with a given prefix.
[string]: Each line of this string will be prefixed.
[prefix]: The string to put in front of each line.
returns: All lines of the input string, each of them prefixed with the given prefix.

tcolors

Defines some tput related constants. In order to change terminal colors you can then use something such as

```
echo "$(tput setaf ${B_TCOLORS[red]})This is red, \
$(tput setaf ${B_TCOLORS[blue]})this blue, $(tput sgr0)this normal."
```

tput can do a lot more than colors, see: man tput & man terminfo.

0.1

Dependencies

tput

Imports

no imports

Global Variables

B_TCOLORS *tcolors/B_TCOLORS*
Global map for human readable colors to tput style color identifiers.
Currently supported values: black|red|green|yellow|blue|magenta|cyan|white

traps

Collection of trap related functions.

0.2

Dependencies

no dependencies

Imports

no imports

Functions

b_traps_getCodeFor [signal] *traps/b_traps_getCodeFor*
Retrieve the current trap code / commands for the given signal.
returns: The current code and sets a zero exit code on success.
@B_E

b_traps_add [code] [signal] [tag] [append flag] *traps/b_traps_add*
Add the given commands to the given trap signal.
[code]: Whatever should be added to the trap.
[signal]: Name of the signal to add the commands to.
[tag]: An optional *unique* marker for these commands so that they can be removed with b_traps_remove later on.
[append flag]: Whether to append the new commands to the end (0: default) or insert them in the beginning (1).
returns: Whatever the internal call to *trap* to set the new trap returns.
@B_E

b_traps_prepend [code] [signal] [tag] *traps/b_traps_prepend*
Prepend the given commands to the ones currently existing for the given trap signal.

Convenience wrapper to b_traps_add with [append flag] set to 1.
[code]: see b_traps_add
[signal]: see b_traps_add
[tag]: see b_traps_add

returns: see b_traps_add
@B_E

b_traps_remove [signal] [tag] *traps/b_traps_remove*
Remove the commands tagged with the given tag from the signal trap.
[signal]: Name of the signal to remove the commands from.
[tag]: The *unique* marker to identify the commands to be removed.
returns: Nothing, but sets a zero exit code on success. May error out if the tag isn't found or the internal trap call failed.
@B_E

types

Functions for data type checks and conversions.

0.6

Dependencies

python3

Imports

no imports

Global Variables

B_TYPES_ENCODING *types/B_TYPES_ENCODING*
String encoding parameter for b_types_str and b_types_parseString. Default: ascii
`ascii` makes sense in 99% of all cases as scripts should use ASCII only anyway (when no user-interaction is involved) in order to remain portable. Keep in mind that bash also needs to support the target encoding in order to support further processing. Currently available encodings: https://docs.python.org/3.7/library/codecs.html#standard-encodings

B_TYPES_MAX_BYTES *types/B_TYPES_MAX_BYTES*
Parameter for b_types_str and b_types_parseString to specify the maximum number of bytes to read per line (default: -1/infinite). Ignore additional bytes. Useful to avoid memory DoS for untrusted input.

B_TYPES_CHECK_NON_BINARY *types/B_TYPES_CHECK_NON_BINARY*
Whether or not b_types_int should also ensure that the command output is non-binary (default: 0/true).

Functions

b_types_parseString [encoding] [max bytes] *types/b_types_parseString*
Checks whether whatever is lying in stdin is a string (and not binary) and if so, prints it to stdout.

Important:

- bash has major issues whenever binary data is involved. For example equality checks may return undefined results. So whenever you are unsure as to whether a variable is a string or not, better pass it thorugh this function.
- The input is taken from *stdin* rather than as parameter as binary parameters may also cause issues (special bytes etc.).
- Even builtins such as `echo` do not necessarily play well with binary data. So it is recommended to pipe binary data through this function before further processing.

Examples:

```
#check a file
b_types_parseString < "/path/to/potential/binary" > /dev/null && echo "It is a string file.

#read parts of a file as string
str="$(dd if="/path/to/another/file" bs=1 skip=8 | b_types_parseString)"
[ $? -eq 0 ] && echo "Found the following string: $str"
```

[encoding]: The encoding of the string lying in stdin. Default: B_TYPES_ENCODING/ascii
[max bytes]: Maximum number of bytes to read per line. Additional bytes are ignored. Default: B_TYPES_MAX_BYTES/-1/infinite
returns: The data as String, if the input data was found to be a String. If no String was found to be lying in stdin, the output is an undefined string and a non-zero exit code is set. B_E is only called on exceptional errors.
@B_E

b_types_str [cmd] [cmd arg 1] ... [cmd arg n] *types/b_types_str*
Execute the given command and ensure that its output / stdout is a string.
Use B_TYPES_ENCODING to specify the encoding (default: ascii) and B_TYPES_MAX_BYTES (default: -1/infinite) to specify the maximum number of bytes to read.
[cmd]: Command to execute. Stdout is checked to be a string, stderr is passed through.
[cmd arg i]: Arguments to pass to [cmd].
returns: Stdout of [cmd], if it is a string and an undefined string otherwise. The exit code is that of the executed command. B_E is called if the output is *not* a string.
@B_E

b__types__parseInteger *types/b_types_parseInteger*
Checks whether whatever is lying in stdin is an Integer and if so, prints it to stdout.
B_TYPES_CHECK_NON_BINARY can be used to enable/disable additional non-binary checking (default: 0/enabled).
returns: The data as Integer, if the input data was found to be an Integer. If no Integer was found to be lying in stdin, the output is an undefined string and a non-zero exit code is set. B_E is only called on exceptional errors.
@B_E

b__types__isInteger [string] *types/b_types_isInteger*
Check whether the given String is an integer (positive or negative) or not.
B_TYPES_CHECK_NON_BINARY can be used to enable/disable additional non-binary checking (default: 0/enabled).
[string]: The string to check.
returns: Nothing, but sets a zero exit code if and only if the given string represents an integer.

b__types__assertInteger [string] [error msg] *types/b_types_assertInteger*
Check whether the given String is an integer (positive or negative) and if not, error out.
B_TYPES_CHECK_NON_BINARY can be used to enable/disable additional non-binary checking (default: 0/enabled).
[string]: The string to check.
[error msg]: Error message to use, if the check fails (optional).
returns: Nothing. If it's no integer, B_E is called.
@B_E

b__types__int [cmd] [cmd arg 1] ... [cmd arg n] *types/b_types_int*
Execute the given command and ensure that its output / stdout is an integer (positive or negative).
B_TYPES_CHECK_NON_BINARY can be used to enable/disable additional non-binary checking (default: 0/enabled).
[cmd]: Command to execute. Stdout is checked to be an integer, stderr is passed through.
[cmd arg i]: Arguments to pass to [cmd].
returns: Stdout of [cmd], if it is an integer and an undefined string otherwise. The exit code is that of the executed command. B_E is called if the output is *not* an integer.
@B_E

b__types__looksLikeArray [string] *types/b_types_looksLikeArray*
Check whether the given String "looks like" a bash array or not.
It may still contain malicious code or whatnot. So you **must not** rely on this function when processing untrusted input.

[string]: The string to check. Use `"$(declare -p var)"` on variables to obtain it.
returns: Nothing, but sets a zero exit code if the given string looks like it could be `eval`'ed to a bash array. References (`declare -n`) will result in a non-zero exit code.

b__types__assertLooksLikeArray [string] [error msg] *types/b_types_assertLooksLikeArray*
Check whether the given String "looks like" a bash array and if not, error out.
It may still contain malicious code or whatnot. So you **must not** rely on this function when processing untrusted input.
[string]: The string to check. Use `"$(declare -p var)"` on variables to obtain it.
[error msg]: Error message to use, if the check fails (optional).
returns: Nothing. If doesn't look like a bash array, B__E is called.
@B__E

b__types__looksLikeMap [string] *types/b_types_looksLikeMap*
Check whether the given String "looks like" a bash map / associative array or not.
It may still contain malicious code or whatnot. So you **must not** rely on this function when processing untrusted input.
[string]: The string to check. Use `"$(declare -p var)"` on variables to obtain it.
returns: Nothing, but sets a zero exit code if the given string looks like it could be `eval`'ed to a bash associative array. References (`declare -n`) will result in a non-zero exit code.

b__types__assertLooksLikeMap [string] [error msg] *types/b_types_assertLooksLikeMap*
Check whether the given String "looks like" a bash map / associative array and if not, error out.
It may still contain malicious code or whatnot. So you **must not** rely on this function when processing untrusted input.
[string]: The string to check. Use `"$(declare -p var)"` on variables to obtain it.
[error msg]: Error message to use, if the check fails (optional).
returns: Nothing. If doesn't look like a bash array, B__E is called.
@B__E

ui

Collection of user interface and user interaction related functions.

0.2

Dependencies

no dependencies

Imports

no imports

Functions

b__ui__passwordPrompt [output var] [ui mode] [prompt string] *ui/b_ui_passwordPrompt*
Prompt the user for a password. Should run inside the parent process (i.e. not inside a subshell).
Once you do not need the password anymore, it is recommended to wipe it from memory as such:

```
#overwrite the password in memory with zeroes, then free it
pass="${pass//?/0}" ; pass=""
```

[output var]: The name of the variable to write the password to.
[ui mode]: How to request the password from the user: auto|gui|tty (default: auto).
[prompt string]: The string to present to the user asking for the password (default: "Password:").
returns: Nothing. Sets a non-zero exit code on errors.
@B_E

wm

Collection of functions related to window managers.

Copyright (C) 2020 David Hobach LGPLv3
0.1

Dependencies

wmctrl

Imports

no imports

Functions

b__wm__getActiveWindowProperties [variable name] *wm/b_wm_getActiveWindowProperties*
Retrieve information about the currently active windows.
[variable name]: Name of the associative array to use as output.
returns: A string specifying an associative array in bash syntax. You can eval

that string to obtain all relevant data. On errors B_E is called.
The information is returned as an associative array containing the following data:
[window ID]_[id] = hexadecimal window ID
[window ID]_[desktop] = desktop number
[window ID]_[pid] = PID of the window
[window ID]_[x] = x-offset
[window ID]_[y] = y-offset
[window ID]_[width] = window width
[window ID]_[height] = window height
[window ID]_[class] = window class name
[window ID]_[client] = client machine name
[window ID]_[title] = window title
Missing data may result in an empty String for the respective property.
@B_E

tests/test_common.bash

Some code and vars meant to be shared across bats tests.

Copyright (C) 2018 David Hobach LGPLv3
0.5

Dependencies

no dependencies

Imports

no imports

Global Variables

TEST_STATE *tests/test_common.bash/TEST_STATE*
A map which can be used to create a persistent state across multiple tests.
By default, bats creates a new shell environment for each test it runs, resetting all changes to global variables.
The state can be managed with the load/save/clearBlibTestState functions below.

Functions

loadBlib *tests/test_common.bash/loadBlib*
Loads blib for testing.
returns: Nothing

skipIfNoUserData *tests/test_common.bash/skipIfNoUserData*
Skip the test if no user data was found.
returns: Nothing.

skipIfCommandMissing [command] *tests/test_common.bash/skipIfCommandMissing*
Skip the test if the given command is not installed.
returns: Nothing.

skipIfNotQubesDom0 *tests/test_common.bash/skipIfNotQubesDom0*
Skip the test if we're not running inside Qubes OS dom0.
returns: Nothing.

skipIfNotRoot *tests/test_common.bash/skipIfNotRoot*
Skip the test if root is not available.
returns: Nothing.

loadBlibTestState *tests/test_common.bash/loadBlibTestState*
Load the TEST_STATE with the data that was saved last via saveBlibState. If you want to use TEST_STATE, call this function during test setup.
returns: Nothing.

saveBlibTestState *tests/test_common.bash/saveBlibTestState*
Save the current TEST_STATE to make it available for further tests.
returns: Nothing.

clearBlibTestState *tests/test_common.bash/clearBlibTestState*
Clears the current test state and removes its persistent files.
returns: Nothing.

testGetterSetter [setter function] [value to set] [reset] *tests/test_common.bash/testGetterSetter*
Executes the given setter function *in the current environment* and makes sure the respective getter function (assumed to have the same name with just a *get* instead of *set*) returns that value.
[setter function]: name of the setter function to call
[value to set]: value to set with the setter function
[reset]: if set to 0 (default), reset the value back to its original value after testing the setter function
returns: nothing, but errors out on test failures

startTimer *tests/test_common.bash/startTimer*
Start a new time measurement window.
returns: nothing

endTimer *tests/test_common.bash/endTimer*
Get the differencee in time in seconds since the last time startTimer was called.
returns: time difference in seconds

funcTimeout [timeout] [function] args *tests/test_common.bash/funcTimeout*
Run the given function with a timeout inside a subshell.
[timeout]: Timeout in seconds after which to terminate the function.
[function]: The function to execute.
[args]: Function arguments.
returns: An exit code of 124, if the function timed out. Otherwise returns whatever the function returned.

runSL [commands] *tests/test_common.bash/runSL*
A version of the bats `run` command which makes sure that the bash runtime state does not change after running the given commands (SL = stateless).
This *should* be the default method of executing tests for blib.
If the given commands are expected to change the state, use runSC instead. In particular the default bats `run` should almost never be used.
Also prints an identifier for easier debugging. The identifier starts at 1 on the first run call per test and increases with each further run call.
WARNING: As the bats `run` it runs inside a subshell. So don't expect changes to persist.
[commands]: The commands to run.
returns: whatever bats run returns

runSC [commands] *tests/test_common.bash/runSC*
A version of the bats `run` command which ignore changes to the bash runtime state (SC = state changing).
If the given commands are expected to be stateless, use runSL instead. In particular the default bats `run` should almost never be used.
Also prints an identifier for easier debugging. The identifier starts at 1 on the first run call per test and increases with each further run call.
WARNING: As the bats `run` it runs inside a subshell. So don't expect changes to persist.
[commands]: The commands to run.
returns: whatever bats run returns

assertReadOnly [path] *tests/test_common.bash/assertReadOnly*
Assert that the given file or directory is read-only by attempting to write to it.
[path]: Full path to a file or directory. If it is not r/o, it may be changed.
returns: A zero exit code, if the path is r/o.
@B_E

tests/user_test_data.bash

Example of a user_test_data file.
That file is meant for static bash varibles that must be set *by the user* (as they may e.g. be OS specific) in order for some tests to work.
If that file doesn't exist or could not be loaded, these tests may be skipped by blib.

0.5

Dependencies

no dependencies

Imports

no imports

Global Variables

UTD_ONLINE *tests/user_test_data.bash/UTD_ONLINE*
Whether or not the test machine is connected to the Internet.

UTD_OS *tests/user_test_data.bash/UTD_OS*
Specify your operating system here, currently supported values:
debian|fedora|red hat|centos|ubuntu|opensuse|qubes dom0|other
Qubes VMs should be specified with their OS.

UTD_QUBES *tests/user_test_data.bash/UTD_QUBES*
Specify whether you are running Qubes OS and in what environment here, possible values:
no|vm|dom0

UTD_QUBES_TESTVM *tests/user_test_data.bash/UTD_QUBES_TESTVM*
If you're using Qubes OS, please specify a *disposable* virtual machine with a static name to be used for testing here. This test VM may crash, be destroyed or whatever - so please don't use a production VM!
Apart from that, the tests will be conducted with disposable VMs with dynamic names using your default template.

Example command to create such a test VM:

```
qvm-create --class DispVM --prop netvm='' --template nonet-dvm -l red d-testing
```

Can safely be ignored if you don't run Qubes OS-related tests.

UTD_QUBES_TESTVM_PERSISTENT *tests/user_test_data.bash/UTD_QUBES_TESTVM_PERS*
If you're using Qubes OS, please specify a *persistent/non-disposable* virtual machine with a static name to be used for testing here. This test VM may crash, be destroyed or whatever - so please don't use a production VM!

Example command to create such a test VM:

```
qvm-create -l red --prop netvm='' testing-pers
```

Can safely be ignored if you don't run Qubes OS-related tests.

UTD_QUBES_DISPVM_TEMPLATE *tests/user_test_data.bash/UTD_QUBES_DISPVM_TEMPLA*
Qubes OS disposable VM template to use for testing. The test code will create disposable VMs from that template. It will *not* modify the template itself.
If no template is specified, the Qubes OS default template is used.

UTD_PW_FREE_USER *tests/user_test_data.bash/UTD_PW_FREE_USER*
A user that allows password-less logons with sudo or su for testing.

If you run bats as root, neither sudo nor su should ask for a password.
Tests requiring admin privileges (e.g. mounting devices) may require this to be root or skip otherwise.

Reference List

assertReadOnly

B_ARGS

b_args_assertOptions

b_args_get

b_args_getAll

b_args_getAllOptions

b_args_getCount

b_args_getInt

b_args_getOption

b_args_getOptionCount

b_args_getOptionInt

b_args_getOptionParamSeparator

b_args_init

B_ARGS_OPTS

b_args_parse

b_args_setOptionParamSeparator

b_arr_contains

b_arr_join

b_arr_mapsAreEqual

b_arr_toList

b_assertLastPipes

b_cachingMessageHandler

B_CALLER_NAME

b_cdoc_cbPrintFirstParam

b_cdoc_cbPrintNewline

b_cdoc_generate

b_cdoc_generateBlibStyle

b_cdoc_getBlockCallback

b_cdoc_getDocumentBeginCallback

b_cdoc_getDocumentEndCallback

b_cdoc_getExtractionRegex

b_cdoc_getFileCallback

b_cdoc_getPostProcessingCallback

b_cdoc_setBlockCallback

b_cdoc_setDocumentBeginCallback

b_cdoc_setDocumentEndCallback

b_cdoc_setExtractionRegex

b_cdoc_setFileCallback

b_cdoc_setPostProcessingCallback

b_checkVersion

B_CONF_DIR

b_daemon_getPid

B_DAEMON_ID

b_daemon_init

b_daemon_restart

b_daemon_sendSignal

b_daemon_start

b_daemon_status

b_daemon_statusPid

b_daemon_stop

b_date_add

b_date_addDays

b_date_diff

b_date_getFileModAge

b_defaultErrorHandler

b_defaultMessageHandler

b_delay_execute

B_DELAY_EXECUTED

b_delay_getCommandAt

b_delay_to

b_deps

b_dmcrypt_close

b_dmcrypt_createLuks

b_dmcrypt_getMapperName

b_dmcrypt_init

b_dmcrypt_isOpen

b_dmcrypt_open

b_dom0_applyFirewall

b_dom0_attachDevice

b_dom0_attachFile

b_dom0_attachVMDisk

b_dom0_clearFirewall

b_dom0_closeCrypt

b_dom0_copy

b_dom0_createLoopDeviceIfNecessary

b_dom0_crossAttachDevice

b_dom0_crossAttachFile

b_dom0_crossCopy

b_dom0_detachDevice

b_dom0_disconnectEventLoop

b_dom0_ensureHalted

b_dom0_ensureRunning

b_dom0_enterEventLoop

b_dom0_execFuncIn

b_dom0_execIn

b_dom0_execStrIn

b_dom0_exists

b_dom0_getDefaultPoolDriver

b_dom0_getDispVMs

b_dom0_getPoolDriver

b_dom0_getQvmBlockInfo

b_dom0_getVMDeps

b_dom0_isHalted

b_dom0_isMountedIn

b_dom0_isRunning

b_dom0_mountIfNecessary

b_dom0_openCrypt

b_dom0_parseQvmBlock

b_dom0_qvmRun

B_DOM0_QVM_RUN_PARAMS

b_dom0_removeUnusedLoopDevice

b_dom0_setVMDeps

b_dom0_startDispVM

b_dom0_testMultiple

B_DOM0_VM_STDERR

b_dom0_waitForFileIn

B_E

b_enforceUser

B_ERR

b_error

b_execFuncAs

b_execFuncInCurrentContext

b_fd_closeAll

b_fd_closeNonStandard

b_fd_getOpen

b_flog_close

b_flog_defaultHeader

b_flog_getDateFormat

b_flog_getHeaderFunction

b_flog_getLogReductionLinesLowerBound

b_flog_getLogReductionLinesUpperBound

b_flog_headerDateScriptSeverity

b_flog_headerDateSeverity

b_flog_init

b_flog_log

b_flog_messageHandler

b_flog_printSeverity

b_flog_setDateFormat

b_flog_setHeaderFunction

b_flog_setLogReductionLinesApprox

b_flog_setLogReductionLinesLowerBound

b_flog_setLogReductionLinesUpperBound

B_FLOG_SEV

b_fs_createLoopDeviceIfNecessary

b_fs_enumerate

b_fs_getBlockDevice

b_fs_getLineCount

b_fs_getMountpoints

b_fs_isEmptyDir

b_fs_isRotatingDrive

b_fs_mountIfNecessary

b_fs_parseSize

b_fs_removeRelativelySafely

b_fs_removeUnusedLoopDevice

b_fs_removeWithOverwrite

b_fs_waitForFile

b_generateStandalone

b_getBlibModules

b_getDefaultMessageHandlerIntermediate

b_getDefaultMessageHandlerPrefix

b_getErrorHandler

b_getMessageHandler

b_hash_file

b_hash_str

B_HTTP_CHECKURLS

b_http_getOnlineStatus

b_http_rawUrlDecode

b_http_rawUrlEncode

b_http_testProxy

b_import

b_info

b_ini_assertNames

b_ini_get

b_ini_getBool

b_ini_getInt

b_ini_getString

b_ini_read

b_initCachingMessageHandler

b_ipcm_change

b_ipcm_get

b_ipcm_getNamespace

b_ipcm_setNamespace

b_ipcm_unsetNamespace

b_ipcv_load

b_ipcv_loadNamespace

b_ipcv_save

b_ipcv_unset

b_ipcv_unsetNamespace

b_isFunction

b_isModule

b_keys_add

b_keys_close

b_keys_delete

b_keys_get

b_keys_getAll

b_keys_getContent

b_keys_getDefaultStore

b_keys_init

B_LIB_DIR

b_listContains

b_meta_getClearDeps

b_meta_getClearImports

b_mtx_create

b_mtx_forceRelease

b_mtx_getSleepTime

b_mtx_pass

b_mtx_release

b_mtx_setSleepTime

b_mtx_try

b_mtx_waitFor

b_multiw_getMaxHangTime

b_multiw_remove

b_multiw_setMaxHangTime

b_multiw_write

B_NET_CHECKHOSTS

B_NET_DNSSERVERS

b_net_getDNSStatus

b_nop

b_notify_send

b_notify_sendNoError

b_notify_waitForUserDbus

b_osid_init

b_osid_isCentOS

b_osid_isDebian

b_osid_isDebianLike

b_osid_isFedora

b_osid_isFedoraLike

b_osid_isOpenSuse

b_osid_isQubesDom0

b_osid_isQubesVM

b_osid_isRedHat

b_osid_isUbuntu

b_printStackTrace

b_proc_childExists

b_proc_killAndWait

b_proc_killByRegexAndWait

b_proc_pidExists

b_proc_resolveSignal

b_proc_waitForPid

B_RC

b_resetErrorHandler

B_SCRIPT

B_SCRIPT_DIR

B_SCRIPT_NAME

b_setBE

b_setDefaultMessageHandlerIntermediate

b_setDefaultMessageHandlerPrefix

b_setErrorHandler

b_setMessageHandler

b_silence

b_sqlite3_close

b_sqlite3_exec

b_sqlite3_getOpen

b_sqlite3_open

b_str_prefixLines

b_str_stripQuotes

b_str_trim

B_TCOLORS

B_TEST_MODE

b_traps_add

b_traps_getCodeFor

b_traps_prepend

b_traps_remove

b_types_assertInteger

b_types_assertLooksLikeArray

b_types_assertLooksLikeMap

B_TYPES_CHECK_NON_BINARY

B_TYPES_ENCODING

b_types_int

b_types_isInteger

b_types_looksLikeArray

b_types_looksLikeMap

B_TYPES_MAX_BYTES

b_types_parseInteger

b_types_parseString

b_types_str

b_ui_passwordPrompt

b_version

b_wm_getActiveWindowProperties

clearBlibTestState

endTimer

funcTimeout

loadBlib

loadBlibTestState

runSC

runSL

saveBlibTestState

skipIfCommandMissing

skipIfNotQubesDom0

skipIfNotRoot

skipIfNoUserData

startTimer

testGetterSetter

TEST_STATE

UTD_ONLINE

UTD_OS

UTD_PW_FREE_USER

UTD_QUBES

UTD_QUBES_DISPVM_TEMPLATE

UTD_QUBES_TESTVM

UTD_QUBES_TESTVM_PERSISTENT