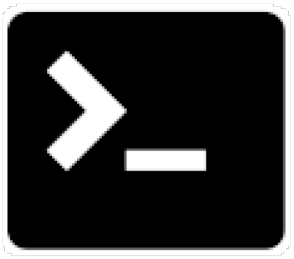
Alinex Bash Library

Helper programs and libraries for administration tool development (mostly in bash)

Alexander Schilling

Table of contents

1.1 Bash Library



This is a collection of helper programs and libraries for administration tool development (mostly in bash). The complete library is build into combined packages, which mostly can be added as single include file to work.

This documentation will show you how to use them.

1.1.1 Installation

Mostly you won't install the bash-lib on a server but create some scripts based on it. But that won't mean you can't or you shouldn't do so. As always it depends on your needs.

See the following chapter for all the possibilities to use it.

First copy the full distribution package or only the parts you need standalone or within your application onto a server. See the downloads page for the latest releases.

Standalone Installation

For a standalone installation move the complete distribution to the to a central position and use it from there. This will often be in `/opt/bash-lib` and should also be included in the path for the standalone commands to be used:

```
wget https://alinex.gitlab.io/bash-lib/downloads/bash-lib.tgz
tar -xzvf bash-lib.tgz
cp dist /opt/bash-lib
echo "PATH=\"$PATH:/opt/bash-lib\"" >> ~/.bashrc
```

But you can also always copy the command directly to any other folder and use it from there.

Some of the commands needs additional helpers, which are not always installed on your server. To check if you have everything working run the checker once directly on your server:

```
curl https://alinex.gitlab.io/bash-lib/downloads/install | bash
```

This will give you information what is needed and if possible automatically installs the needed packages.

Including libraries

The most common way is to include the needed library from the release packages within the destination code. That ensures that changes to the bash-lib repository or a central installation won't change the running code. You can update it yourself anytime you want to by overwriting it with a newer version.

As already said, all files are self contained without further references. So copy the needed library as `lib.bash` to your project and include it relatively:

```
source_dir=$(dirname $(readlink -f "${BASH_SOURCE[0]:-$(pwd)/x}"))
source "$source_dir/lib.bash"
```

You can also pick the files directly from the complete distribution or add all of them, it's very small.

Short usage

To only use it in single bash script, put the library directly beside your script:

```
source_dir=$(dirname $(readlink -f "${BASH_SOURCE[0]:-$(pwd)/x}"))
source "$source_dir/base.bash"

# here you can use it
```

See also the following Skeleton which gives you a fast start for your own script.

Install and Use

If you want to run a script with the bash lib but don't know if it is already installed, you can always check and install before you use it automatically:

```
# load bash lib or use alternative log method
log() { echo $2; }
source_dir=$(dirname $(readlink -f "${BASH_SOURCE[0]:-$(pwd)/x}"))
if [ -e "$source_dir/lib/base.bash" ]; then
    source "$source_dir/lib/base.bash"
else
    # update bash lib
    log INFO "Installing bash-lib..."
    rm -rf "$source_dir/lib"
    curl -s https://alinex.gitlab.io/bash-lib/downloads/bash-lib.tgz | tar -xz
    mv dist "$source_dir/lib"
    source "$source_dir/lib/base.bash"
fi
```

And if you want to always update it to the newest version use:

```
# load bash lib or use alternative log method
log() { echo $2; }
source_dir=$(dirname $(readlink -f "${BASH_SOURCE[0]:-$(pwd)/x}"))
if [ -e "$source_dir/lib/base.bash" ]; then
    source "$source_dir/lib/base.bash"
fi
# update bash lib
log INFO "Updating bash-lib..."
rm -rf "$source_dir/lib"
curl -s https://alinex.gitlab.io/bash-lib/downloads/bash-lib.tgz | tar -xz
mv dist "$source_dir/lib"
source "$source_dir/lib/base.bash"
```

1.1.2 Statistics

The following statistics will give you a better understanding about it's complexity. The documentation has 34 pages (in A4 PDF format) and contains 41030 characters.

1.1.3 Download Documentation

If you want to have an offline access to the documentation, feel free to download the 34 pages PDF Documentation.

1.1.4 License

Alinex Bash Library

Apache License, Version 2.0

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

SendMail (Perl)

This utility is included as separate helper tool.

Originally written as SendEmail by: Brandon Zehm caspian@dotconf.net under GPL http://caspian.dotconf.net/menu/Software/SendEmail (version 1.56, Sep 29th 2009)

Modified 2018 by Alexander Schilling 2018 to support environment variables for default settings.

GPLv2

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

www.gnu.org/licenses/old-licenses/gpl-2.0-standalone.html.

Last update: January 17, 2023

1.2 Skeleton

The skeleton is used as a base to make your own script. Use it as template, copy it and fill in your specific program parts.

The most common parts like configuration, bash-lib loading and options checker are already included.

```
#!/bin/bash

# Script to....
#
# Usage: skeleton <params>
#
# The skeleton is used as a base to make your own scripts.

# Include library
source_dir=$(dirname $(readlink -f "${BASH_SOURCE[0]:-$(pwd)/x}"))
source "$source_dir/../lib/base.bash"

# Configuration
LOG_LEVEL=DEBUG
LOG_FILE=${LOG_FILE:-$(path)/../log/$(basename $0).log}

# If you also work in gitlab with release tags
VERSION=$(git describe --abbrev=0 --tags 2>/dev/null)
VERSION=${VERSION:-master} # use master if not defined
REVISION=$(git rev-parse --verify master)
GITLAB=$(git config --get remote.origin.url | sed 's/\.git$//')

# Parameter checking
parsed=$(getopt --options=:Vh --longoptions=version,help --name "$0" -- "$@")
[ ${PIPESTATUS[0]} -eq 0 ] || log_exit "Could not parse parameters!"
eval set -- "$parsed"
while true; do
    case "$1" in
        -V|--version) echo "$(basename $0) version $VERSION"; exit 0 ;;
        -h|--help) source "$(path)/$(basename $0).help"; exit 0 ;;
        --) shift; break ;;
        *) break ;;
    esac
done
[ $# -gt 0 ] \
&& log WARN "No parameters are supported, only -h to show help - ignoring them"

# change into command directory
cd "$(path)"

# Processing




log NOTICE Done.
```

Copy this code and change it to your specification and add the processing...

Last update: January 17, 2023

1.3 Last Changes

1.3.1 Version 1.4.1 (2021-01-21)

- Fix to use LOG_LEVEL_DEFAULT
- Fix bug with wrong use of basename in log module
- Update doc theme

1.3.2 Version 1.4.0 (2020-03-08)

- Updated documentation
- Fixed bug in setting exit code in lock_exit
- added status in middleware detection
- added more info methods for current load and memory usage
- fixed logging if called without terminal (ssh)

1.3.3 Version 1.3.1 (22.01.2020)

- command logging fixed (had problems if called through sudo or cron)
- error detection optimized in logging
- Optimized logging within lock/unlock
- Support the TRACE level in logging to be more detailed as DEBUG

1.3.4 Version 1.3.0 (05.07.2019)

- support piped input in `psql_exec`
- increase info package with
    - hardware detection
    - geo location
    - package analyzation
    - user and crontab analyzation
- support for more linux OS versions in detection
- automatic unit testing based on shunit2
- add sudo support to core

1.3.5 Version 1.2.1 (21.02.2019)

- fix process locking in `lock`
- add changelog
- add readme, license and changelog to distribution

1.3.6 Version 1.2.0 (29.12.2018)

- made repository public available
- add MkDocs based website
- restructure documentation
- move skeleton to documentation
- add clear license information
- add downloads for distribution files

1.3.7 Version 1.1.0 (07.12.2018)

- compress multiline commands
- update documentation

- update online help

1.3.8 Version 1.0.3 (05.12.2018)

- remove character conversion for psql_csv
- fix lock_exit to really work
- log started steps in process
- update process documentation

1.3.9 Version 1.0.2 (02.12.2018)

- change code style

1.3.10 Version 1.0.1 (28.11.2018)

- update documentation
- fix OS detection for Debian
- fix csv -> html conversion for psql
- fix psql error checking
- add core lib for path

1.3.11 Version 1.0.0 (25.11.2018)

- color management
- log handler
- locking management
- psql integration
- info module

Last update: January 17, 2023

1.4 Roadmap

Currently nothing is planned, here.

Last update: January 17, 2023

1.5 Privacy statement

This documentation is part of the **alinex.gitlab.io** site and as such please have a look on the site's privacy statement.

Last update: January 17, 2023

2.1 Libraries

Methods may return two different parts:

- `$?` exit code which is 0 on success
- direct console output which may be captured or be piped (internal commands will directly call the log module so no need to do this here)

The following modules are available:

- core general small helpers
- colors colorization methods
- log log handler to write to file, `STDERR` or syslog
- process serialize or parallelize tasks
- psql to access PostgreSQL Database
- info is a collection of system information methods

Last update: January 17, 2023

2.2 Core

This part of the bash lib contains some general helper methods which are not suitable to be put in other lib parts.

2.2.1 path

Get the absolute path to the called script. It's the same as `$source_dir` which is used in the general examples but this stays the same while the variable may be changed in scripts.

```
# load the help file shich will output help pages directly beside the script
source "$(path)/$(basename $0).help"
```

It is always the directory path of the first called script which included the libraries.

2.2.2 usesudo

This will return a short string containing `'sudo'` or an empty string `''` if no `sudo` is needed or possible. This will return the sudo command only if not user `root` and `sudo` is generally allowed for the active user.

This can be stored in a variable `$usesudo` and used to make command calls:

```
usesudo=$(usesudo)
$usesudo systemctl start my_service
```

2.2.3 needsudo

This method will check the `sudo` rights and exit with an alert message if there is a problem. This can be:

- if the active user is not `root` and `sudo` is not allowed
- if commands are given aren't allowed to be called using `sudo` for the active user

To check only the general `sudo` need and allowance:

```
needsudo
```

And to check some specific commands use:

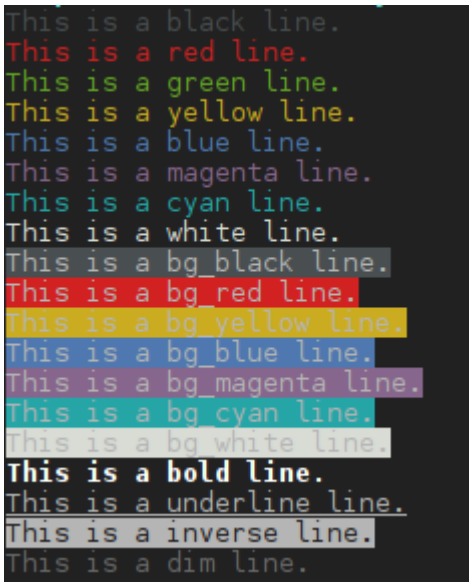
```
needsudo "systemctl start tomcat8"
```

Last update: January 17, 2023

2.3 Color Output

Setup color methods to be used in bash scripts for formatting. It contains methods to output ANSI color codes.

The possible colors and styles supported are:



The terminal will automatically interpret this but depending on the color scheme of your terminal the colors may differ.

To also see the color codes in other programs use:

- `alias less='less -R'` enables ANSI color support

2.3.1 Include

First you have to load the library or any package, because all contain them:

```
source src/bash-lib/colors.bash  # load color methods
source dist/core.bash            # or load the core package
```

See the general usage on how to include.

2.3.2 Add color or style

To set the different colors and styles you have different methods, which all work the same way:

Format text if a text is given as arguments, use style only for this

```
red "Direct ouput without newline"
echo "$(red 'output with newline') but not this"
echo "$(red multiple parameters are joined by spaces)"
x=$(red "load colorized into variable")
x="manually switch $(red +)on$(reset) and off"
```

Above you see the different possibilities to use. While line 1 to 3 will write to `STDOUT` the last two lines will store the colored output in a variable.

Text given as pipe instead of parameters

Like done in bash with other methods, you can also pipe the stream to colorize it:

```
echo "This text is colored using pipe" | cyan
```


But then no parameter is allowed to the color or style method.

Separately start and end style if called with `+` or without arguments and pipe

```
red +
echo -n "Text in red"
bold +
echo -n " and bold"
reset
echo " and normal again.
```

As shown above you start using the `+` sign and end all styles with `reset`. The `-n` switch to echo is used to prevent newline on the first two calls and get all into one line.

2.3.3 Defined colors and styles

Foreground color

The following methods will set the foreground colors:

- `black`
- `red`
- `green`
- `yellow`
- `blue`
- `magenta`
- `cyan`
- `white`

Background color

The following methods will set the background colors:

- `bg_black`
- `bg_red`
- `bg_green`
- `bg_yellow`
- `bg_blue`
- `bg_magenta`
- `bg_cyan`
- `bg_white`

Styles

The following methods will set other styles:

- `bold`
- `underline`
- `inverse`
- `dim`

2.3.4 Remove colors and styles

And if you want to remove the coloring later again use the `decolor` method:

```
ctext=$(bg_red "This text with red background")
echo "original: $ctext"
echo "decolor using parameters: $(decolor $ctext)"
echo "decolor using pipe: $ctext" | decolor
```


2.3.5 Convert to HTML

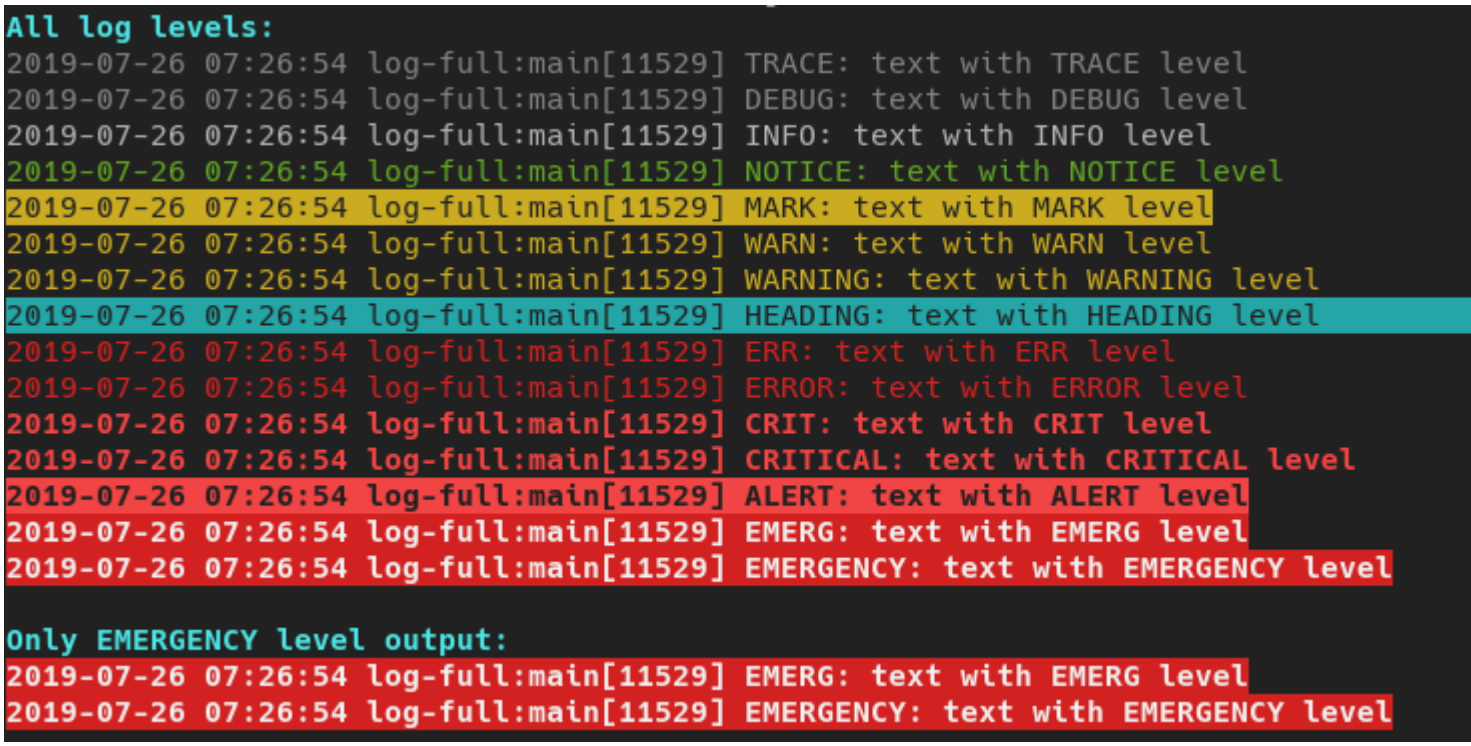
This is easily done using aha, a small converter which may be installed as linux package from the default repository.

```
command 2>&1 | aha -b | mail -s "Command Report" -a "Content-Type: text/html" myself@email.de
```

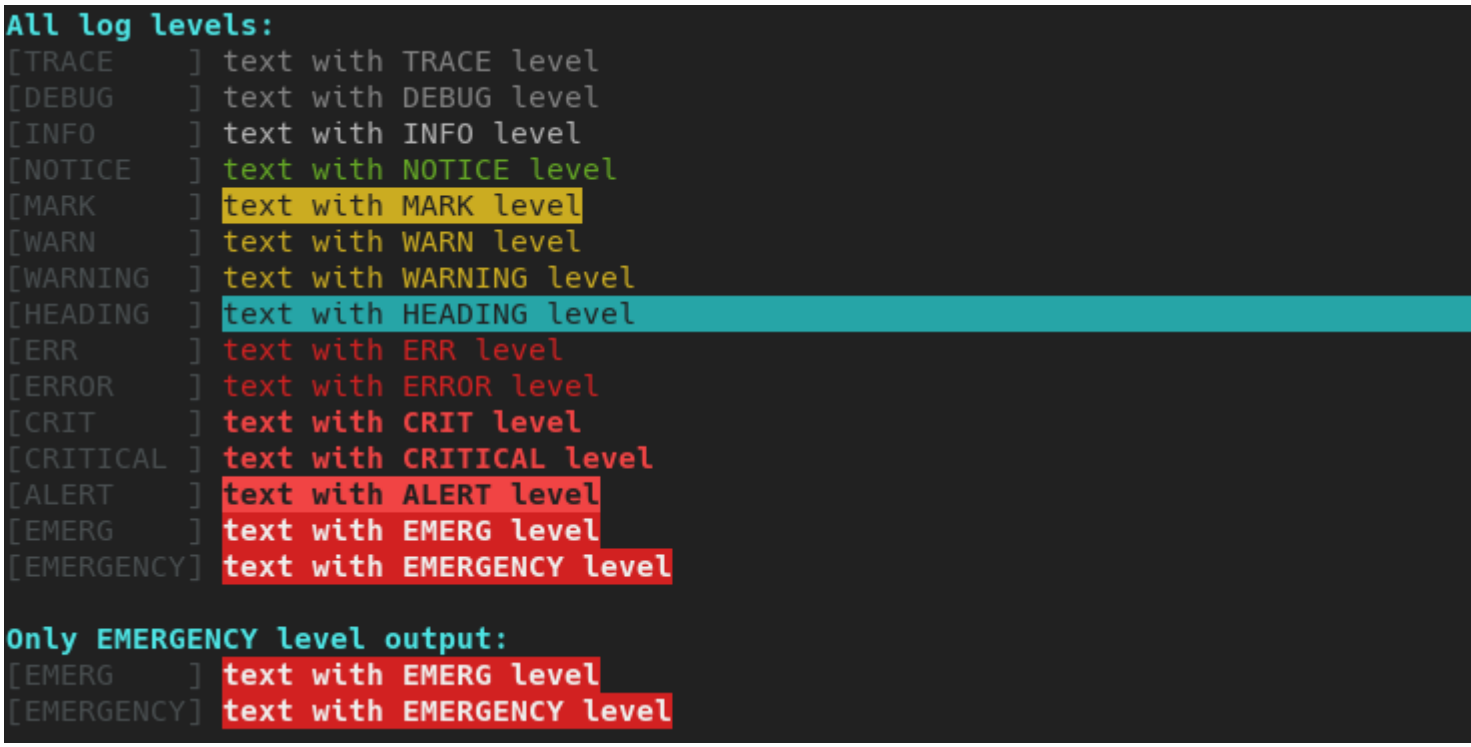
Last update: January 17, 2023

2.4 Log Handler

A handler to write logs with an easy to use logging library that can be sourced from scripts. It allows logging to an arbitrary file, to `STDERR`, or to a syslog facility. It supports eight logging levels.



The log handler can be used as program or library.



With the `SIMPLE` console output for `STDERR` it can also be used as colorization toolkit. `STDOUT` is not supported here because it may become problematic with functions doing both, sending results through `STDOUT` and logging to `STDOUT`.

2.4.1 Usage

Setup

For usage as command or library the configuration is the same and fully optional:

```
# basic output selection
LOG_CONSOLE='SIMPLE'                # output to STDERR (default)
LOG_CONSOLE='FULL'                  # output to STDERR without date, tag, pid and type
# alternatively or additionally use one of the following
LOG_FILE='/var/log/myscript.log'    # output in file
SYSLOG_FACILITY='local7'            # output to syslog
# specify logging
LOG_LEVEL='INFO'                    # minimum log level
LOG_LEVEL_DEFAULT='AUTO_INFO'       # default log level to use if none
LOG_DATE_FORMAT="+%Y-%m-%d %H:%M:%S"
LOG_TAG="test"                      # defaults to current process name
# file rotation
LOG_ROTATE_TIME=[DAILY|WEEKLY|MONTHLY]
LOG_ROTATE_SIZE=<bytes>
LOG_ROTATE_NUM=<max number of files>
LOG_ROTATE_COMPRESS=1
# user defined auto detection and coloring (regexp)
LOG_AUTO[OK]="\b(done|transferred)\b"
LOG_AUTO[MARK]="!!!"
# command log calls
LOG_CMD_QUIET=1                     # will prevent call and success messages
LOG_CMD_LEVEL=AUTO_INFO             # the log level used for command output
```

Command

To use it within the shell you can use the binary under `bin/log`. For easier use you may also add it to the path like used in the following example:

```
log <type> <message>      # log to file
cat xxx | log             # pipe to log
cat xxx | log <type>      # pipe with specific log type
log INFO <../list.txt     # output file contents
```

Library

If you use it within another bash file you can also include the library and use it directly, which will also include the colors library:

```
source ../bash-lib/log.bash  # log handler
```

After that messages may be invoked easily using:

```
log INFO "process is working"
log_exit EMERG "preprocessing not done, stopping" 16
```

While the first call will only output the log message, the second call also exits the running program with the additionally given exit code.

To log the call of some other routines use:

```
log_cmd date +%Y-%m-%d
```

This will use the auto detection logger and give you the result of the command in variable `$result` while also piping the output together with errors to the log module.

2.4.2 Piping messages

But you can also pipe output from other commands directly to the log:

```
run-process | log INFO # log stdin
run-process 2>&1 >/dev/null | log ERROR # log stderr
run-process |& log INFO # log stdin + stderr
result=$(run-process |& tee >(log) | cat) # log output and store it in variable
```

The following lines show how to use different settings for `STDOUT` and `STDERR`.


```
( run-process | log INFO ) 3>&1 1>&2 2>&3 | log ERROR # log both differently
( run-process 3>&1 1>&2 2>&3 | log ERROR ) 3>&1 1>&2 2>&3 | log INFO # priorize INFO
```

But keep in mind that this may lead to double logging if `LOG_CONSOLE` is used.

You can also use `tee` to duplicate output streams.

2.4.3 Auto detect Level

Often useful in pipes but also usable in other log messages is the special `AUTO` log setting:

```
run-process |& log
run-process |& log AUTO
```

This will auto detect the concrete log level for each line. Currently `TRACE`, `DEBUG`, `INFO`, `NOTICE`, `MARK`, `WARN`, `WARNING`, `HEADING`, `ERR`, `ERROR`, `CRIT`, `CRITICAL`, `ALERT`, `EMERG` and `EMERGENCY` will trigger the specified log type. Some other keywords are also interpreted and all other lines are output using the minimum level.

You can also specify a higher or lower minimum level as `DEBUG` by using:

```
run-process |& log AUTO_INFO
run-process |& log AUTO_WARN
run-process |& log AUTO_TRACE
```

If this is set the minimum level be the given one but it will be increased by auto detection.

To add more rules for the auto detection you may add a regular expression per each log level:

```
declare -A LOG_AUTO # only needed if defined before loading the liubrary
LOG_AUTO[OK]="\b(done|transferred)\b"
LOG_AUTO[MARK]="!!!"
```

It is always case insensitive and will be used additionaly to the default detection.

2.4.4 Log Levels

Eight logging levels are supported, combining the levels from the Python logging module and RFC 5424.

Level	Numeric	Syslog	Origin	Usage
TRACE or VERBOSE	5	7	Log Utilities	Very detailed logging (not always used)
DEBUG	10	7	RFC 5424	Diagnostically helpful messages
INFO	20	6	RFC 5424	Something which may be useful to know
NOTICE	25	5	RFC 5424	Success message or step done
MARK	25	5	own extension	Special marked like information asked for
WARN or WARNING	30	4	RFC 5424	Warning which may be OK
HEADING	35	4	own extension	Start of new bigger Part
ERR or ERROR	40	3	RFC 5424	An error which can occur
CRIT or CRITICAL	50	2	RFC 5424	An error which should not occur
ALERT	60	1	RFC 5424	Very critical like incorrect method call
EMERG or EMERGENCY	70	0	RFC 5424	Something which should never happen

Setting the `LOG_LEVEL` in the script will log subsequent log messages at that value or higher only. The `LOG_LEVEL` may be changed anytime within the script.

While the most log levels are from the error levels, we also added `MARK` and `HEADING` as report levels which don't have an real error case. This allows to also use the log library to send informational higher prioritized output.

2.4.5 File rotation

While the library keeps the log file opened for better performance you can't rotate it using external tools. But the integrated rotation will do perfectly fine.

But you are also free to do this on your own.

Rotate by date

```
LOG_ROTATE_TIME=DAILY   # date as YYYY-MM-DD
LOG_ROTATE_TIME=WEEKLY  # date as YYYY_week_WW
LOG_ROTATE_TIME=MONTHLY # date as YYYY-MM
```

If this is set the current logs will go in the normal log file but on a new day the old file will be renamed with it's date pattern appended.

The rotated files may also be compressed by setting the `LOG_ROTATE_COMPRESS` flag.

Rotate by size

To rotate on fixed file size use:

```
LOG_ROTATE_SIZE=<bytes>
LOG_ROTATE_NUM=<max number of files>
```


The rotated files may also be compressed by setting the `LOG_ROTATE_COMPRESS` flag.

But you can't combine the two rotation methods by date and by size, currently.

2.4.6 Switching log file

If really necessary, it is possible to switch log file by changing the setting of `LOG_FILE` and calling `log_init` without parameters.

2.4.7 Log Commands

It is also possible to log details from the commands completely:

```
log_cmd sudo apt-get update >/dev/null
```

This will log the called command, the output and error messages, the exit code and pipe the standard output further on.

```
[INFO     ] calling: sudo apt-get update
[TRACE    ] OK:1 http://de.archive.ubuntu.com/ubuntu bionic InRelease
[TRACE    ] OK:2 http://linux.teamviewer.com/deb stable InRelease
[TRACE    ] OK:3 http://ppa.launchpad.net/micahflee/ppa/ubuntu bionic InRelease
[TRACE    ] OK:4 http://de.archive.ubuntu.com/ubuntu bionic-updates InRelease
[TRACE    ] Holen:5 http://archive.neon.kde.org/user bionic InRelease [131 kB]
[TRACE    ] OK:6 http://apt.postgresql.org/pub/repos/apt buster-pgdg InRelease
[TRACE    ] OK:7 http://packages.microsoft.com/repos/vscode stable InRelease
[TRACE    ] OK:8 http://de.archive.ubuntu.com/ubuntu bionic-backports InRelease
[TRACE    ] OK:9 http://security.ubuntu.com/ubuntu bionic-security InRelease
[TRACE    ] OK:10 http://apt.postgresql.org/pub/repos/apt jessie-pgdg InRelease
[TRACE    ] OK:11 https://repo.fortinet.com/repo/ubuntu /bionic InRelease
[TRACE    ] Es wurden 131 kB in 2 s geholt (76,0 kB/s).
[TRACE    ] Paketlisten werden gelesen...
[WARN     ] W: Das Laden der konfigurierten Datei »multiverse/binary-i386/Packages« wird übersprungen, da das Depot »https://repo.fortinet.com/repo/ubuntu /bior
[NOTICE   ] sudo call succeeded
```

Last update: January 17, 2023

2.5 System Information

This is a collection of methods to gain system information.

2.5.1 Base Information

The general information is stored within variables on load:

- `OS` - type of OS like: Linux, windows, mac, Solaris, AIX
- `KERNEL` - version number like: 4.4.0-135-generic
- `MACH` - machine type like: x86_64
- `DIST_BASE` - base distribution (for Linux): RedHat, SuSe, Mandrake, Debian
- `DIST_BASE_REV` - base distribution number or name (if possible)
- `DIST` - distribution like: LinuxMint, Ubuntu
- `REV` - revision number of distribution
- `REV_NAME` - code name of this revision

You can use these directly!

But to get them all combined in a human readable line call:

```
info=$(system_info)
echo $info # Linux system with kernel 4.15.0-38-generic x86_64 (neon 18.04 bionic based on Debian)
```

2.5.2 Extended System Info

This information is reachable with some parameter less function calls.

hw_machine_id

Unique ID for the hardware machine if defined.

hw_virtual

Returns `true` if this is a virtual machine.

hw_cores

Returns the number of cpu cores available.

hw_cores

Returns the number of cpu cores available.

hw_load

Returns the short load of the system.

hw_processor

Returns the processor model name which may include some technical specifications.

hw_memory_mb

Returns the memory size in MB.

hw_free_mb

Returns the free memory size in MB.

hw_avail_mb

Returns the available memory, which is available for processes. This may be occupied by caching at the moment.

hw_swap_mb

Returns the swap size in MB.

hw_swap_free_mb

Returns the free swap size in MB.

hw_disks

Returns a list of disks with size, usage (percentage) and mount point.

ip_main

Returns the main IP address.

ip_list

Returns all listening IP addresses.

ip_info

Retrieves some geo location information from the IP the system uses to enter the public internet like:

```
ip 46.237.195.215
hostname HSI-KBW-46-237-195-215.hsi.kabel-badenwuerttemberg.de
city Dornhan
region Baden-Württemberg Region
country DE
loc 48.3501,8.5090
postal 72175
org AS29562 Unitymedia BW GmbH
```

2.5.3 Packages

package

With `package` you may check if a specific package is installed and get the version from it.

```
tomcat=$(package tomcat) || log WARN "No tomcat installed!"
echo $tomcat # will output 8
```

As seen above you can also use shortcuts which don't completely equal to the package name because it will be extended automatically as far as this is predefined in code.

package_list

You can also check which of the special systems (hard coded list) are installed. It will return: `<package name> <version>`

2.5.4 Configuration

ssh_keys

If called this script will generate a list of all allowed ssh-key based users with: `<account> <name> <type> <key>`

sudoers

This will create a list with all users on the system and what they are allowed to call using sudo: `<account> <sudo-rights>`

cron_tasks

This will go through:

- user crontabs
- cron.d scripts
- hourly, daily, weekly, monthly tasks

All found entries will be combined into the following format: `<where> <minute> <hour> <day> <month> <wday> <user> <command>`.

Last update: January 17, 2023

2.6 Process Serial/Parallel

This methods will help you to lock a special process that it can't run in parallel using file based locks. If the same lock is used another time in another process or sub process it will wait till the first one releases the flag. You have to give a lock file path to be used as flag.

Another part allows to simplify parallel tasks which may be subroutines or commands.

2.6.1 Include

First you have to include this helper in your bash script:

```
source ../bash-lib/process.bash # load functions
```

After that you can use one of the following methods.

2.6.2 Locking

In any part of your script you can surround a block with `lock` and `unlock` statements. The process will wait on the `lock` statement till no other process with the same lock is running before going on.

Function: lock

Used to set the lock or wait and set it, if already locked.

Parameter:

- full path to use as lock file which should be writable (default to `LOCKFILE`)

Function: unlock

Used to remove the lock.

Parameter:

- full path to use as lock file which should be writable (default to `LOCKFILE`)

A simple locking will look like:

```
lockfile=/tmp/my-program-lock
lock $lockfile   # create the lock
# here comes the critical code
# which will run only once at a time
unlock $lockfile # remove the lock
```

As an alternative `lock_exit` can be used to abort if this is already locked. It won't wait till it can get the lock but exit immediately.

Function: lock_exit

Try to set the lock, if not possible exit the whole process there with an error message.

Parameter:

- full path to use as lock file which should be writable (default to `LOCKFILE`)
- error-message (default: `Stop processing because this is locked in $lockfile`)
- error-code (default is `1`)

```
lock_exit $lockfile $message $code  # ... and exit if already locked
```

If you don't give a lockfile the environment variable `LOCKFILE` or `tmp/<process-name>-lock` will be used.

Configuration

The only configurable value here beside the logging is:

```
LOCKFILE="tmp/$(basename $0)-lock" # use name of current process
LOCK_SLEEP=10 # time to wait till retrying if other process locked it
```

Checking the Lock

The locking is done by local files whose name part is given or used from the current running script. so `mx-program` will work like:

1. The `lock` is set by making a file containing the filename with the PID as file extension and content. This indicates, that this PID is waiting to retrieve the lock like `/tmp/my-program-lock.1587`
2. remove lockfile `/tmp/my-program-lock` if the process within is no longer active
3. Create a softlink without extension for it `/tmp/my-program-lock -> /tmp/my-program-lock.1587` if there is already such an softlink, try again every second.
4. Remove the softlink and the lock with the PID on `unlock`

If the program is terminated in between some old files may be present. The code also contains a `trap` to prevent such problems by removing them also on breaks.

While waiting to get the lock the process is:

1. wait the defined time `$LOCK_SLEEP`
2. if the PID file `/tmp/my-program-lock.1587` got lost recreate it
3. remove lockfile if the process within is no longer active
4. retry creating the softlink

You can always remove all the lock files by hand if the PID is no longer running. But this should neither be needed.

2.6.3 Asynchronous Calls

Running some tasks in parallel can save time but may be problematic to manage. This functions help to simplify this tasks.

Function: async

Run the given command asynchronous ang go on in the calling routine.

Parameter:

- command to execute, which may be a subroutine or real shell command
- optional arguments for the command

Function: async_name

Alternative to `async` in which this call is given a name to refer in `async_wait`.

Parameter:

- any unique identifier
- command to execute, which may be a subroutine or real shell command
- optional arguments for the command

Function: async_wait

Check if the command is done or wait here till it is so.

Parameter:

- identifier from `async_name` or the command from `async` call
- function to call if a failure occurred within the process

A simple call will be:

```
f1() {
  sleep 5
  echo "done f1 with value $1 (=155)"
}

async f1 155
echo "comes first"
async_wait f1
echo "done"
```

The function `f1` is called asynchronously and while this runs the following echo statement will be called. `async_wait`

If the same command or function is used multiple times in parallel use `async_name` which let`s you also define an individual identifier:

```
async_name f1_1 f1 155
async_wait f1_1
```

Additionally you can give a function name as second parameter to `async_wait` which will be called with an error message if the asynchronous function returned an error code. See at step control below for an example.

To wait for all asynchronous processes to end use only `wait`, but then you won't get the individual return codes.

Configuration

The possible configuration beside logging is:

```
LOCK_SLEEP=10 # time to wait before rechecking for the lock
LOCKFILE="/tmp/$(basename $0)-lock" # lockfile used if none given in function call
STEPFILE="/tmp/$(basename $0)-steps" # for async steps (see below)
```

Step Control

If a `STEPFILE` is defined each step will be checked if is already done (entry in the file). Only if not done it will be started. So a process which is canceled within can be processed further on.

The stepfile will look like:

```
Finished: step f1 at 2018-11-15 09:54
Finished: step f2 at 2018-11-15 09:56
```

But if the process is finished, the file will be removed.

A complete integration may look like:

```
# Configuration
STEPFILE=${STEPFILE:-/tmp/$(basename $0)-steps}
WORK_DATE=${WORK_DATE:-$(date +%Y-%m-%d)}

lock_exit # exit if same process is already running

# Remove old stepfile (not from the current running date)
[ -e $STEPFILE ] \
&& [ "$(date -d "$(stat -c %y $STEPFILE)" +%Y-%m-%d)" != "$WORK_DATE" ] \
&& rm $STEPFILE

# Log message if continue of old run
if [ -e $STEPFILE ]; then
  log WARN "A previously aborted process for $WORK_DATE is found, trying to continue after:"
  cat $STEPFILE | log INFO
```



```
fi

# If process is stopped, also stop subprocesses
trap "trap - SIGTERM && kill -- -$$" SIGINT SIGTERM EXIT

# Write start to stepfile
echo "Starting process at $(date '+%Y-%m-%d %H:%M:%S')" >>$STEPFILE

failed() {
  log_exit ALERT "$1"
}

# Call the steps
async f1
async f2
async_wait f1 failed
async_wait f2 failed

# Cleanup
rm $STEPFILE # clear steps
trap - SIGINT SIGTERM EXIT # remove trap
unlock # free the file lock
```

Last update: January 17, 2023

2.7 PostgreSQL Access

Library for easy database integration in bash scripts.

2.7.1 Configuration

The following environment settings should be used:

- `PGUSER` PostgreSQL user name to connect as
- `PGPASSWORD` password to be used if the server demands password authentication
- `PGHOST` specifies the name of host to connect to
- `PGPORT` port number to connect to at the server host (if not default port 5432)
- `PGDATABASE` is the database name
- `PGLOG` set this flag to log the calls to postgreSQL and the returned results

2.7.2 Methods

psql_exit

Check that the database connection is configured and exit with error message if not. A log message is emitted.

psql_exec or | psql_exec

Run the given query and return only the results:

```
result=$(psql_exec <query>) || log_exit ALERT "Failed to export from DB"
```

Another alternative is to pipe the sql commands into the process:

```
cat file.sql | psql_exec
```

The resulting text may contain the `UPDATE` or `DELETE` success message or in case of `SELECT` it will contain the resulting rows with `|` as record separator.

psql_csv

Run the given query and return the dataset as CSV including the column header:

```
psql_csv <query> >result.csv || log_exit ALERT "Failed to export from DB"
```

This may create a file with the following contents:

```
id,status,changetime,description,reporter,story_priority,resolution,time,component,remaining_time,summary,priority,keywords,drp_resources,milestone,owner,rd_poi
34,accepted,2014-10-09 15:44:50.550308+00:00,,user,,,2014-10-01 16:54:17.745176+00:00,UT-Editor,4,some text,Alpha,,,Scrum Startup,user,,2nd Try,task
39,new,2014-10-09 18:58:44.694814+00:00,"""hello""",user,,,2014-10-01 17:15:45.326223+00:00,Document Management,4,"""integration, test for DocumentService""",Al
43,new,2014-10-09 21:08:34.888938+00:00,"Content:
- Something
Todo:
- Finally test on Beta",user,,,2014-10-02 15:34:05.008107+00:00,,,new Booking Process 1.1: Test on BetaServer,,,,Scrum Startup,user,5,2nd Try,story
```

As shown in the example above special cases are:

- Record 2: containing quotes as content
- Record 3: contains a multiline text field

csv2html or | csv2html

A small helper which will convert a data export like from `psql_csv` into an HTML table:


```
echo "<html><body>" >>result.html
psql_csv <query> | csv2html >>result.html || log_exit ALERT "Failed to export from DB"
echo "</body></html>" >>result.html
```

csv2xls or | csv2xls

Because opening CSV in Microsoft Excel correctly is not an easy task, better create the binary format:

```
psql_csv <query> | csv2xls result.xls || log_exit ALERT "Failed to export from DB"
```

Keep in mind that the `install` command of the bash-lib have to be run to make this work.

Last update: January 17, 2023

3.1 Design Decisions

The bash library is build on a flat architecture. Some libraries needs each other and will be combined together on build.

3.1.1 Directory Structure

The following folders are used:

```
bin             # development helper commands
src             # source code:
    bin         # - commands
    include     # - libraries
test            # test methods
docs            # documentation
dist            # distribution folder
site            # created documentation
```

And the core configuration files are:

```
README.md       # short info
mkdocs.yml      # config for documentation
.gitlab-ci.yml  # continuous integration setup
```

3.1.2 File Structure

The library files have the same structure.

First the necessary base libraries should be loaded. For direct execution this should be done from the current directory which is retrieved in `source_dir`:

```
source_dir=$(dirname $(readlink -f "${BASH_SOURCE[0]:-$(pwd)/x}"))
source "$source_dir/colors.bash" # load color methods
```

Keep in mind that `source_dir` may change within a library and both lines will also be removed while building the packages. If you need the directory later better use the method `$(path)` which is in the base library and gives you the same path.

The next part always is the configuration section. This includes:

- definition of internal variables
- checking of preset environment settings

After maybe some one time pre-processing is done the real methods will follow.

Methods

Each method should at first check all parameters for valid arguments and store the results.

On misconfiguration, improper calls or system problems the command should be stopped using `log_exit ALERT <message>`.

The results will be written as text to `STDOUT`. This allows to pipe them into file or other commands. The exit code have to be `0` if everything succeeds.

3.1.3 Languages

While the bash library is mainly written in bash itself, some more complex operations are done by perl scripts which includes seamlessly into the library.

3.1.4 System Tools

Some of the default unix tools are used like:

- `grep` to match lines
- `tail` and `head` to extract part of the strings
- `sed`, `awk` or `perl` to match or replace parts
- and a lot of other default linux utils

But this should neither be a problem because all of them are base tools and already installed.

Last update: January 17, 2023

3.2 Build Process

To generate the compressed and combined packages a build process is available. By calling `bin/build` from within the project directory the distribution will be build. This will:

- check that everything is checked in and pushed to master
- build libraries
- add header
- remove comments and unnecessary white-space
- remove library includes (not needed in combined packages)
- set `VERSION`, `REVISION`, `GITLAB` to fixed values
- pack them together by concatenation
- build commands
- add header
- remove comments and unnecessary white-space in bash files
- include libraries in bash files
- set `VERSION`, `REVISION`, `GITLAB` to fixed values

The result will be stored in the distribution folder `dist`. This folder will hold the files you may use later.

The `dist` folder contains the combined libraries, described below:

```
-rw-rw-r-- 1 alex alex  9994 Nov  9 08:11 base.bash
-rw-rw-r-- 1 alex alex 10970 Nov  9 08:11 psql.bash
-rw-rw-r-- 1 alex alex 11603 Nov  9 08:11 locking.bash
-rw-rw-r-- 1 alex alex 13350 Nov  9 08:11 info.bash
-rw-rw-r-- 1 alex alex 15935 Nov  9 08:11 all.bash
```

And also some commands (without extension):

```
-rwxrwxr-x 1 alex alex   549 Nov  9 08:23 log
-rwxrwxr-x 1 alex alex 79640 Nov  9 08:23 sendmail
```

All files are self contained, so you only pick the ones you need. They are also slightly compressed for better load time.

3.2.1 Setup

Which libraries to build into which packages is configured within the build script:

```
lib[base]="core, colors, log"
lib[process]="core, colors, log, process"
lib[info]="core, colors, log, info"
lib[psql]="core, colors, log, psql"
lib[all]="core, colors, log, process, info, psql"
```

While the associative array `lib` contains all packages, each package contains a comma separated list of libraries.

To include a library within a command put a comment exactly like below in the code:

```
source_dir=$(dirname $(readlink -f "${BASH_SOURCE[0]:-$(pwd)/x}"))

# DIST include base.bash
source "$source_dir/../include/log.bash" # load log handler
```

That will directly include the `base.bash` at the position of this comment and remove the line behind which is the `source` line.

Variable replacement

The following variables will be replaced by their values in build process:

In bash:

- `VERSION=...`
- `REVISION=...`
- `GITLAB=...`

In Perl:

- `###VERSION###`
- `###REVISION###`
- `###GITLAB###`

Last update: January 17, 2023

4.1 Downloads

The optimized and compressed files are available as:

- Complete distribution (~35kB archive)

Each of the combined and compressed following packages are self contained as a single bash script with some of the libraries:

- Base libraries including core, colors, log
- Info libraries including core, colors, log, info
- Process libraries including core, colors, log, process
- PSQL libraries including core, colors, log, psql
- All libraries including core, colors, log, locking, info, psql

You should always only include one library. If you need multiple functions, include a bigger one.

All of the above files are build from latest master branch automatically.

Also you may need the following script to check if your server is ready for the bash-lib or if you should install something:

- Install checker

Last update: January 17, 2023

4.2 License

4.2.1 Alinex Bash Library

4.2.2 SendMail (Perl)

This utility is included as separate helper tool.

Last update: January 17, 2023

