Bashinator

Bash Shell Script Framework

Bashinator

Contents

- Introduction & Features
- Screenshots
- Function usage examples
- File system layouts of...
  - ...a typical Bashinator installation
  - ...a typical application installation
- Example application script
- Build your own application
- Questions & Feedback`

Bashinator

Introduction & Features

Bashinator is a framework for bash shell scripts.

Its features include:

- Flexible and powerful message handling: print, log (to syslog and/or logfile) and mail messages (configureable based on severity) with a single message function, for example:
  - Log messages from debug...alert, print messages from info...alert, mail messages from warning...alert
  - Log and mail messages with timestamps, print messages without timestamps
  - Log to syslog and logfile at once, overwrite or append to the logfile
- Colored output (if printed to a terminal supporting colors, redirected output is not colored)
- Exhaustive information (timestamps, message severity, source file, line numbers and function names) in messages to ease debugging
- Function call stack traces
- Logging of sub-command output (capturing stdout/stderr of external commands to a dedicated temporary logfile)
- Lockfile handling

Bashinator

Screenshots, part 1

```
[wschlich@prometheus(pts/50):example]$ ./example.sh -a
[2009-05-27 21:35:43 +02:00] >>> [____DEBUG] {example.lib.sh:49}, __init(): command line argument: -a
[2009-05-27 21:35:44 +02:00] >>> [____DEBUG] {bashinator.lib.0.sh:151}, __prepare(): successfully created temporary script subcommand logfile '/tmp/example.log.6DUeOY'
[2009-05-27 21:35:44 +02:00] >>> [____DEBUG] {bashinator.lib.0.sh:159}, __prepare(): script subcommand logfile: '/tmp/example.log.6DUeOY'
[2009-05-27 21:35:44 +02:00] >>> [____DEBUG] {example.lib.sh:76}, __main(): this is a debug test
[2009-05-27 21:35:44 +02:00] >>> [_____INFO] {example.lib.sh:76}, __main(): this is a info test
[2009-05-27 21:35:44 +02:00] >>> [___NOTICE] {example.lib.sh:76}, __main(): this is a notice test
[2009-05-27 21:35:44 +02:00] !!! [__WARNING] {example.lib.sh:76}, __main(): this is a warning test
[2009-05-27 21:35:44 +02:00] !!! [____ERROR] {example.lib.sh:76}, __main(): this is a err test
[2009-05-27 21:35:44 +02:00] !!! [_CRITICAL] {example.lib.sh:76}, __main(): this is a crit test
[2009-05-27 21:35:44 +02:00] !!! [____ALERT] {example.lib.sh:76}, __main(): this is a alert test
[2009-05-27 21:35:44 +02:00] !!! [EMERGENCY] {example.lib.sh:76}, __main(): this is a emerg test
[2009-05-27 21:35:44 +02:00] >>> [____DEBUG] {example.lib.sh:117}, exampleFunction(): fooArgument: foo
[2009-05-27 21:35:44 +02:00] >>> [____DEBUG] {example.lib.sh:120}, exampleFunction(): barArgument: bar
[2009-05-27 21:35:44 +02:00] >>> [_____INFO] {example.lib.sh:124}, exampleFunction(): this is an example function
[2009-05-27 21:35:44 +02:00] !!! [____ALERT] {example.lib.sh:138}, bazFunction(): FATAL: dying for test purposes
[2009-05-27 21:35:44 +02:00] !!! [____ALERT] {example.lib.sh:138}, bazFunction(): function call stack (most recent last):
[2009-05-27 21:35:44 +02:00] !!! [____ALERT] {example.lib.sh:138}, bazFunction(): --> __dispatch('-a') called in 'example.sh' on line 71
[2009-05-27 21:35:44 +02:00] !!! [____ALERT] {example.lib.sh:138}, bazFunction(): --> __main() called in 'bashinator.lib.0.sh' on line 115
[2009-05-27 21:35:44 +02:00] !!! [____ALERT] {example.lib.sh:138}, bazFunction(): --> fooFunction('fooArgs') called in 'example.lib.sh' on line 85
[2009-05-27 21:35:44 +02:00] !!! [____ALERT] {example.lib.sh:138}, bazFunction(): --> barFunction('barArgs') called in 'example.lib.sh' on line 130
[2009-05-27 21:35:44 +02:00] !!! [____ALERT] {example.lib.sh:138}, bazFunction(): --> bazFunction('bazArgs') called in 'example.lib.sh' on line 134
[2009-05-27 21:35:44 +02:00] !!! [____ALERT] {example.lib.sh:138}, bazFunction(): please check script subcommand log '/tmp/example.log.6DUeOY' for details
[2009-05-27 21:35:44 +02:00] !!! [____ALERT] {example.lib.sh:138}, bazFunction(): terminating script
[2009-05-27 21:35:44 +02:00] >>> [____DEBUG] {bashinator.lib.0.sh:775}, __mail(): mailFrom: wschlich <wschlich@prometheus.bla.fasel.org>
[2009-05-27 21:35:44 +02:00] >>> [____DEBUG] {bashinator.lib.0.sh:782}, __mail(): mailEnvelopeFrom: wschlich@prometheus.bla.fasel.org
[2009-05-27 21:35:44 +02:00] >>> [____DEBUG] {bashinator.lib.0.sh:789}, __mail(): mailRecipient: wschlich@prometheus.bla.fasel.org
[2009-05-27 21:35:44 +02:00] >>> [____DEBUG] {bashinator.lib.0.sh:796}, __mail(): mailSubject: Messages from example.sh running on prometheus.bla.fasel.org
[2009-05-27 21:35:44 +02:00] >>> [____DEBUG] {bashinator.lib.0.sh:829}, __mail(): successfully sent mail via sendmail
[2009-05-27 21:35:44 +02:00] >>> [____DEBUG] {bashinator.lib.0.sh:960}, __msgMail(): successfully sent mail
[2009-05-27 21:35:44 +02:00] >>> [____DEBUG] {bashinator.lib.0.sh:1001}, __trapExit(): successfully mailed saved messages
>>> last exit code: 1
[wschlich@prometheus(pts/50):example]$
```

Bashinator example application, terminal output

Bashinator

Screenshots, part 2

```
From: wschlich <wschlich@prometheus.bla.fasel.org>
To: wschlich@prometheus.bla.fasel.org
Subject: Messages from example.sh running on prometheus.bla.fasel.org
Date: Wed, 27 May 2009 21:37:34 +0200 (CEST)

[2009-05-27 21:37:33 +02:00] [____DEBUG] {example.lib.sh:49}, __init(): command line argument: -a
[2009-05-27 21:37:33 +02:00] [____DEBUG] {bashinator.lib.0.sh:151}, __prepare(): successfully created temporary script subcommand logfile '/tmp/example.log.8hGzmB'
[2009-05-27 21:37:33 +02:00] [____DEBUG] {bashinator.lib.0.sh:159}, __prepare(): script subcommand logfile: '/tmp/example.log.8hGzmB'
[2009-05-27 21:37:33 +02:00] [____DEBUG] {example.lib.sh:76}, __main(): this is a debug test
[2009-05-27 21:37:33 +02:00] [_____INFO] {example.lib.sh:76}, __main(): this is a info test
[2009-05-27 21:37:33 +02:00] [___NOTICE] {example.lib.sh:76}, __main(): this is a notice test
[2009-05-27 21:37:33 +02:00] [__WARNING] {example.lib.sh:76}, __main(): this is a warning test
[2009-05-27 21:37:33 +02:00] [____ERROR] {example.lib.sh:76}, __main(): this is a err test
[2009-05-27 21:37:33 +02:00] [_CRITICAL] {example.lib.sh:76}, __main(): this is a crit test
[2009-05-27 21:37:33 +02:00] [____ALERT] {example.lib.sh:76}, __main(): this is a alert test
[2009-05-27 21:37:33 +02:00] [EMERGENCY] {example.lib.sh:76}, __main(): this is a emerg test
[2009-05-27 21:37:33 +02:00] [____DEBUG] {example.lib.sh:117}, exampleFunction(): fooArgument: foo
[2009-05-27 21:37:33 +02:00] [____DEBUG] {example.lib.sh:120}, exampleFunction(): barArgument: bar
[2009-05-27 21:37:33 +02:00] [_____INFO] {example.lib.sh:124}, exampleFunction(): this is an example function
[2009-05-27 21:37:33 +02:00] [____ALERT] {example.lib.sh:138}, bazFunction(): FATAL: dying for test purposes
[2009-05-27 21:37:34 +02:00] [____ALERT] {example.lib.sh:138}, bazFunction(): function call stack (most recent last):
[2009-05-27 21:37:34 +02:00] [____ALERT] {example.lib.sh:138}, bazFunction(): --> __dispatch('-a') called in 'example.sh' on line 71
[2009-05-27 21:37:34 +02:00] [____ALERT] {example.lib.sh:138}, bazFunction(): --> __main() called in 'bashinator.lib.0.sh' on line 115
[2009-05-27 21:37:34 +02:00] [____ALERT] {example.lib.sh:138}, bazFunction(): --> fooFunction('fooArgs') called in 'example.lib.sh' on line 85
[2009-05-27 21:37:34 +02:00] [____ALERT] {example.lib.sh:138}, bazFunction(): --> barFunction('barArgs') called in 'example.lib.sh' on line 130
[2009-05-27 21:37:34 +02:00] [____ALERT] {example.lib.sh:138}, bazFunction(): --> bazFunction('bazArgs') called in 'example.lib.sh' on line 134
[2009-05-27 21:37:34 +02:00] [____ALERT] {example.lib.sh:138}, bazFunction(): please check script subcommand log '/tmp/example.log.8hGzmB' for details
[2009-05-27 21:37:34 +02:00] [____ALERT] {example.lib.sh:138}, bazFunction(): terminating script

--8<--[ start of script subcommand log (/tmp/example.log.8hGzmB) ]--8<--
rm: cannot remove `/does/not/exist': No such file or directory
mkdir: cannot create directory `/does/not/exist': No such file or directory
ls: cannot access /does/not/exist: No such file or directory
--8<--[ end of script subcommand log ]--8<--

--
sent by example.sh running on prometheus.bla.fasel.org at 2009-05-27 21:37:34
```

Bashinator example application, mail output

Bashinator

Conventions

- Functions are named in lowerCamelCase
- Global variables are named in UpperCamelCase
- Local variables are named in lowerCamelCase
- Bashinator functions and global variables begin with a double underscore
- Functions that print their results to stdout are named “printSomething”
- Return codes of functions and exit codes of scripts based on Bashinator:
  - 0: positive check result (for check functions only) or successful task completion (for all other functions)
  - 1: negative check result (for check functions only, undefined for others)
  - 2: error (for any function)
- Variables are always initialized (with type where possible) before use
- Functions shouldn't ever terminate the script execution on their own, instead, they should return with an error return code so the calling function can take appropriate action

```
declare -i __SomeGlobalBashinatorIntegerVariable=0
function __someBashinatorFunction() {
     local -a someLocalBashinatorArrayVariable=()
}

declare -a SomeGlobalApplicationArrayVariable=( “foo” “bar” )
function someApplicationFunction() {
     local -i someLocalApplicationIntegerVariable=1
}
```

Bashinator

Function usage examples

```
function createDirectory() {
     local directory=${1}
     if [[ -z "${directory}" ]]; then
          __msg err "argument 1 (directory) missing"
          return 2 # error
     fi
     __msg debug "directory: ${directory}"
     if ! mkdir -p "${directory}" >>"${_L}" 2>&1; then
          __msg err "failed to create directory '${directory}'"
          return 2 # error
     fi
     return 0 # success
}

if [[ ! -d /somedir ]]; then
     __msg info "/somedir does not exist, trying to create it"
     if ! createDirectory /somedir; then
          __die 2 "failed to create /somedir"
     else
          __msg info "successfully created /somedir"
     fi
fi
```

In this example, all Bashinator functions are marked in red.

Bashinator

File system layout of a typical Bashinator installation

```
## package 'app-shells/bashinator' from 'wschlich' overlay

## bashinator library
/usr/lib/bashinator.lib.0.sh # API version 0, corresponds to bashinator ebuild SLOT

## bashinator example application
/usr/share/doc/bashinator-0.3/example/bashinator.cfg.sh # bashinator configuration
/usr/share/doc/bashinator-0.3/example/example.cfg.sh    # application configuration
/usr/share/doc/bashinator-0.3/example/example.lib.sh    # application library
/usr/share/doc/bashinator-0.3/example/example.sh        # application script
```

Bashinator

File system layout of a typical application installation

```
## application named 'myapplication'
/etc/myapplication/bashinator.cfg.sh    # bashinator configuration
/etc/myapplication/myapplication.cfg.sh # application configuration
/usr/lib/myapplication.lib.sh           # application library
/usr/bin/myapplication.sh               # application script
```

Bashinator

Example application script, part 1

```
##
## bashinator basic variables
##

export __ScriptFile=${0##*/} # example.sh
export __ScriptName=${__ScriptFile%.sh} # example
export __ScriptPath=${0%/*}; __ScriptPath=${__ScriptPath%/} # /path/to/this/script
export __ScriptHost=$(hostname -f) # host.example.com
```

At first we define some basic variables that Bashinator needs to function properly.
They are defined at the very beginning of the script that is to be executed (example.sh in this case).

Bashinator

Example application script, part 2

```
##
## bashinator library and config
##

## system installation
export __BashinatorConfig="/etc/${__ScriptName}/bashinator.cfg.sh"
export __BashinatorLibrary="/usr/lib/bashinator.lib.0.sh" # APIv0
## local installation in dedicated script path
#export __BashinatorConfig="${__ScriptPath}/bashinator.cfg.sh"
#export __BashinatorLibrary="${__ScriptPath}/bashinator.lib.0.sh" # APIv0
if ! source "${__BashinatorConfig}"; then
    echo "!!! FATAL: failed to source bashinator config '${__BashinatorConfig}'" 1>&2
    exit 2
fi
if ! source "${__BashinatorLibrary}"; then
    echo "!!! FATAL: failed to source bashinator library '${__BashinatorLibrary}'" 1>&2
    exit 2
fi
```

Next we set the full paths to the Bashinator config used for the current application (contains message handling settings etc.) and to the Bashinator library that is to be used (containing the Bashinator API version in its filename).

We define the corresponding variables and try to source both files.

Bashinator

Example application script, part 3

```
##
## boot bashinator
##

__boot
```

Now we “boot up” Bashinator.

This procedure checks for the required Bash version, sets some required shell options, defines a safe PATH etc.

If the boot procedure succeeds, we can safely use all other Bashinator functionality in addition to __boot().

Bashinator

Example application script, part 4

```
##
## application library and config
##

## system installation
export ApplicationConfig="/etc/${__ScriptName}/${__ScriptName}.cfg.sh"
export ApplicationLibrary="/usr/lib/${__ScriptName}.lib.sh"
## local installation in dedicated script path
#export ApplicationConfig="${__ScriptPath}/${__ScriptName}.cfg.sh"
#export ApplicationLibrary="${__ScriptPath}/${__ScriptName}.lib.sh"

## include required source files
__requireSource "${ApplicationConfig}"
__requireSource "${ApplicationLibrary}"
```

In this example our application consists of a single configuration file and a single library file containing all application-specific code.
We define the corresponding variables and use the Bashinator function __requireSource() to source both files.

The application-specific code must define the functions __init() and __main() which are being executed by Bashinator later on.
The __init() function shall contain the application initialization code (command line parsing, argument validation etc.) and the __main() function shall contain the main application code.

Bashinator

Example application script, part 5

```
##
## dispatch the application with all command line arguments
##

__dispatch "${@}"
```

At the end of the script, we dispatch the application using the Bashinator function __dispatch().

The __dispatch() function then calls the following functions:

1. The application function __init()
2. The Bashinator function __prepare()
3. The application function __main()
4. The Bashinator function __cleanup()

The Bashinator functions __prepare() and __cleanup() are used for sub command log and lock file handling (if configured accordingly).

The application can simply __die() on errors within __init() and __main().

Bashinator

Build your own application

Base your application on the example from the Bashinator release package.

In this example we place all application files below /opt/myapp.

If you're going to create an ebuild for your application, use the typical locations for a system installation instead (see slide “File system layout of a typical application installation” for details).

```
$ layman -a wschlich
$ emerge bashinator
$ mkdir /opt/myapp
$ cp /usr/share/doc/bashinator-*/example/bashinator.cfg.sh  /opt/myapp/bashinator.cfg.sh
$ cp /usr/share/doc/bashinator-*/example/example.cfg.sh     /opt/myapp/myapp.cfg.sh
$ cp /usr/share/doc/bashinator-*/example/example.lib.sh     /opt/myapp/myapp.lib.sh
$ cp /usr/share/doc/bashinator-*/example/example.sh         /opt/myapp/myapp.sh
```

Bashinator

Questions? Feedback?

Please ask your questions now and/or send me your feedback and suggestions:

mailto:wschlich@gentoo.org
xmpp:wschlich@jabber.laber.fasel.org