easybash / **bash-coding-style-guide** Public

Bash Coding Standards: Coding style guide of bash shell script.

MIT license

1 star **0** forks

Star Notifications

Code Issues Pull requests Actions Projects Wiki Security master

Go to file

terrylinooo First commit ··· on Apr 21, 2020 **2**

View code

README.md

Releases

No releases published

Bash Coding Style Guide

This coding style guide is based on Google Shell Style Guide and we made some changes to it.

Packages

No packages published

- Differences
  - **Indentation** - We suggest 4 spaces, just like common programming languages.
  - **Array** - We add this part to coding style guide for increasing readability.
  - **Heredoc** - Clear explanation of here-document.
  - **Shebang** - `#!/usr/bin/env bash` , not `#!/usr/bin/bash`

Other differences may not motion here, for more details you can check out the Table of Content below.

Table of Content

Shell Files

File Extension

- ✓ Executables should have a .sh extension.
- ✓ Libraries (included files) should have a .sh extension.

```
easybash.sh
```

Shebang

- ✓ Shebang lines must always be `#!/usr/bin/env bash`
  `#!/usr/bin/env` searches `PATH` for `bash` . This allows you to change your `PATH` to get the interpreter without having to edit every file you're working on.

```
#!/usr/bin/env bash
```

Include Files

- ✓ 1. The included files should not have shebang lines.
- ✓ 2. Use `source` instead of `.` to include files. Although `.` is POSIX standard, the

`source` word provides more readability.

Yes

```
source "${EASYBASH_DIR}/inc/functions.sh"
```

No `2`

```
# Use source keyword provides more readability.
. "${EASYBASH_DIR}/inc/functions.sh"
```

Naming Convention

Variable

- ✓ 1. Always use underscore naming convention.
- ✓ 2. No camel-case naming convention.
- ✓ 3. If a variable can be changed from its parent environment, it should be in uppercase.
- ✓ 4. If a variable can be changed from a command line argument, it could be in uppercase, and name it starting with an underscore.
- ✓ 5. A constant variable should be in uppercase, and make it `readonly` .
- ✓ 6. If the value of a variable is `ANSI color code` , it should be in uppercase, and name it starting with the word `COLOR_` .
- ✓ 7. Other varibles should be lowercase.

Yes

```
php_version="7.2"
```

No `2`

```
# No camel-case naming convention.
phpVersion="7.2"
```

Yes

```
readonly PHP_VERSION="7.2"
```

No 5

```
# A constant variable should be readony.
PHP_VERSION="7.2"
```

Yes

```
COLOR_EOF="\e[0m"
COLOR_BLUE="\e[34m"
COLOR_RED="\e[91m"
COLOR_GREEN="\e[92m"
COLOR_WHITE="\e[97m"
COLOR_DARK="\e[90m"
COLOR_BG_BLUE="\e[44m"
COLOR_BG_GREEN="\e[42m"
COLOR_BG_DARK="\e[100m"
```

No 6

```
# Should be in uppercase starting with the word `COLOR_`.

eof_sign="\e[0m"
blue="\e[34m"
red="\e[91m"
green="\e[92m"
white="\e[97m"
dark="\e[90m"
bg_blue="\e[44m"
bg_green="\e[42m"
bg_dark="\e[100m"
```

Function

- ✓ 1. Always use underscore naming convention.
- ✓ 2. No camel-case naming convention.
- ✓ 3. A function name should be lowercase.
- ✓ 4. Braces must be on the same line as the function name.

- ✓ 5. Adding a namespace with `::` provides more readability. (recommended)

Yes

```
func_test() {
   local this_is_local="Local variable"
   echo ${this_is_local}
}

func_test
```

No `4`

```
# Braces must be on the same line as the function name.

func_test()
{
   local this_is_local="Local variable"
   echo ${this_is_local}
}

func_test
```

Recommend

```
# Namespace provides more readability.

func::test() {
   local this_is_local="Local variable"
   echo ${this_is_local}
}

func::test
```

Source Filename

- ✓ Lowercase, with underscores to separate words if needed.

Formating

Indentation

- ✓ 1. Indent 4 spaces for each level of indentation.
- ✓ 2. No Tabs
- ✓ 3. Vertical indentation in *array* is 4 spaces and up to five values per line. (example)

```
if [ "${abc}" == "yes" ]; then
    echo "ok"
fi
```

Lines

- ✓ 1. Maximum line length is 80 characters.
- ✓ 2. No trailing spaces in the end of line.
- ✓ 3. No spaces in empty line.

If you need a string that is longer than 80 characters, either use embedded newlines or here documents if possible. Literal strings that have to be longer than 80 characters and can't sensibly be split are ok, but use common sense if in doubt. It's common to use longer lines if it helps, not hurts readability.

```
# DO use 'here document's.
cat << EOF
I am an exceptionally long
string.
EOF

# Embedded newlines are ok too.
long_string="I am an exceptionally
    long string."
```

Pipelines

If a pipeline don't all fit on one line, it should be split at one pipe segment per line with the pipe on the newline. Indent each of the subsequent lines by four spaces.

```
# All fits on one line
command_1 | command_2

# Long commands
command_1 \
    | command_2 \
    | command_3 \
    | command_4
```

Variables

- ✓ 1. Using the local keyword inside functions prevents problems with global variables.
- ✓ 2. Always quote the value, unless the value is Integer.
- ✓ 3. Always brace-quote the variables when using them, except single character shell specials. (ex. `?` , `*` , `#` , etc..)

Yes

```
php_version="7.2"
```

No `3`

```
# Missing quotes.
php_version=7.2
```

Yes

```
func_test() {
   local this_is_local="Local variable"
   echo ${this_is_local}
}

func_test
```

No `8`

```
# Missing local keyword when defining a varible in a function.

func_test() {
   this_is_local="Local variable"
   echo ${this_is_local}
}

func_test
```

Arrays

- ✓ 1. Vertical indentation is 4 space width and up to 5 values per line.
- ✓ 2. Always quotes the values in the array, unless index.

Yes

```
php_modules=(
    "bcmath"    "bz2"         "cgi"          "cli"         "common"
    "curl"      "dba"         "dev"          "enchant"     "gd"
    "gmp"       "imap"        "interbase"    "intl"        "json"
    "ldap"      "mbstring"    "mysql"        "odbc"        "opcache"
    "pgsql"     "phpdbg"      "pspell"       "readline"    "recode"
    "redis"     "snmp"        "soap"         "sqlite3"     "sybase"
    "tidy"      "xml"         "xmlrpc"       "xsl"         "zip"
)
```

Yes

```
array=(
    "one"          "two"     "three"    "four"    [6]="five"
    [8]="eight"    "nine"    "ten"
)
```

No `2`

```
# Missing quotes in each value.
php_modules=(
    bcmath    bz2         cgi          cli         common
    curl      dba         dev          enchant     gd
    gmp       imap        interbase    intl        json
    ldap      mbstring    mysql        odbc        opcache
    pgsql     phpdbg      pspell       readline    recode
    redis     snmp        soap         sqlite3     sybase
    tidy      xml         xmlrpc       xsl         zip
)
```

No `1` `2`

```
# Missing vertical indentation.
php_modules=(bcmath bz2 cgi cli common curl dba dev enchant gd gmp imap
interbase intl jsonldap mbstring mysql odbc opcache pgsql phpdbg pspell
readline recode redis snmp soap sqlite3 sybase tidy xml xmlrpc xsl zip)
```

No `2`

```
# Missing quotes in each value.
array=(
    one         two     three    four    [6]=five
    [8]=eight   nine    ten
)
```

Heredoc

- ✓ 1. Delimiting identifier should always be uppercase.
- ✓ 2. No semicolon `;` after delimiting identifier.
- ✓ 3. Redirect and the delimiting identifier should be separated by a space.

YES

```
cat << EOF
This is an example.
easybash
EOF
```

No `1`

```
# Delimiting identifier should be uppercase.
cat << eof
This is an example.
easybash
eof
```

No `2`

```
# Semicolon is unnecessary.
cat << EOF;
This is an example.
easybash
EOF
```

No `3`

```
# Missing a sapce between redirect and the delimiting identifier.
cat <<EOF
This is an example.
easybash
EOF
```

Quoting

- ✓ Always quote strings containing variables, command substitutions, spaces or shell meta characters, unless careful unquoted expansion is required.
- ✓ Double quotes is strongly preferred.

Control Structures

If Statement

- ✓ `;` and `then` should be on the same line, separated by a space.
- ✓ `else` 、 `elif` 、 `fi` should be on its own line vertically aligned with the `if` statement.

```
status="error"

# if..fi statement
if [ "${status}" == "success" ]; then
    echo "success"
else
    echo "other"
fi

# if..elif..fi statement
if [ "${status}" == "success" ]; then
    echo "success"
elif [ "${status}" == "error" ]; then
    echo "error"
else
    echo "other"
fi
```

For Statement

- ✓ `;` and `do` should be on the same line, separated by a space.
- ✓ `done` should be on its own line vertically aligned with the `for` statement

```
for module in ${php_modules[@]}; do
    func_easybash_msg info "Proceeding to install PHP module
\"${module}\" ..."
    sudo ${_PM} install -y php${package_version}-${module}
done
```

While Statement

- ✓ `;` and `do` should be on the same line, separated by a space.
- ✓ `done` should be on its own line vertically aligned with the `while` statement.

```
i=0

while [ ${i} -lt 5 ]; do
    i=$(($i+1))
    echo ${i}
done
```

Case Statement

- ✓ Indent 4 spaces.
- ✓ The patterns `"action")` and the corresponding action terminator `;;` are indented at the same level.
- ✓ The pattern strings should be around with double quotes to keep readability.
- ✓ Do not quote pattern-matching metacharacters. (`*` , `?` , `|` , etc...)

```
while [ "$#" -gt 0 ]; do
    case "$1" in
        "-v")
            package_version="${2}"
            shift 2
        ;;
        "--version="*)
            package_version="${1#*=}";
```

```
                shift 1
            ;;
            # Help
            "-h"|"--help")
                show_script_help
                exit 1
            ;;
            # Info
            "-i"|"--information")
                show_script_information
                exit 1
            ;;
            "-"*)
                echo "Unknown option: ${1}"
                exit 1
            ;;
            *)
                echo "Unknown option: ${1}"
                exit 1
            ;;
        esac
    done
```

Comments

Conclusion