Shell Script Best Practice

**TABLE OF CONTENTS**

[Foreword: Hammers and Nails 1](#_Toc875786208)

[Best Practice 1](#_Toc1574650438)

[Usage of Shell Vs Programming Languages 1](#_Toc584244836)

[Shell Choice 2](#_Toc1703169651)

[Styling 2](#_Toc1981140124)

[Styling: Spaces vs Tabs 2](#_Toc210983981)

[Styling: Shell Naming 2](#_Toc1124764803)

[Styling: Shell Variable Naming 2](#_Toc1194990704)

[Styling: Shell Function Naming 3](#_Toc1100320377)

[Styling: Python Naming Side Rules 3](#_Toc1721756629)

[Code Documentation 4](#_Toc724404424)

[TODO Comments 4](#_Toc2103276724)

[Function Comments 4](#_Toc1816445907)

[Script Basics 5](#_Toc455204031)

[Script name 5](#_Toc64082865)

[Security 5](#_Toc1708623822)

[Command Line Arguments 5](#_Toc1276859713)

[Variables Quoting / Usage (there be dragons) 5](#_Toc1352621679)

[Terminal Output 6](#_Toc1251360370)

[Terminal Output: STDOUT STDERR Pipes 6](#_Toc2134334239)

[Terminal Output: Colours 6](#_Toc1371808889)

[Versioning 6](#_Toc23194914)

[Software Versioning 7](#_Toc1637746018)

[Semantic versioning Cliffnotes 7](#_Toc300184004)

[Further Reading 7](#_Toc332904331)

# Foreword: Hammers and Nails

There's an expression "Law of the Instrument"

I suppose it is tempting, if the only tool you have is a hammer, to treat everything as if it were a nail

[*https://en.wikipedia.org/wiki/Law\_of\_the\_instrument*](https://en.wikipedia.org/wiki/Law_of_the_instrument)

Remember this when selecting the right tool for a job

Bash is an excellent scripting language and a poor programming language

Likewise python is an excellent programming language but a poorer\* scripting language

*\* though not poor, see node/java for poor scripting languages*

# Best Practice

## Usage of Shell Vs Programming Languages

* Shell **SHOULD** only be used for small utilities or simple wrapper scripts.
* Shell **SHOULD NOT** be used for complex tasks
* *Aside : once saw a bespoke XML to JSON converter written in pure Bash script, this was neither efficient nor easy to maintain, 1000+ line script was later replaced with 120-130 line node program*

Some google guidelines:

* shell is an excellent choice for calling other utilities and are doing relatively little data manipulation
* when performance matters, use something other than shell
* When a script that is more than a few hundred lines long, or that uses non-straightforward control flow logic, consider rewriting it in a programming language
* When assessing the complexity of your code consider whether the code is easily maintainable by people other than its author.

## Shell Choice

* **SHOULD** use bash over non portable or rarer shells
* *exception for client*
* *customer is always correct, where clients demand or require cmd / powershell based solutions, obviously must provide solution*
* Shell type **MUST** be declared in shebang in first line of executable script
* #!/bin/bash
* **SHOULD** use set command at start of script to configure bash shell
* set -x
* *more information here* [*https://ss64.com/bash/set.html*](https://ss64.com/bash/set.html)

## Styling

Follow google conventions with the follow additions

*see* [*https://google.github.io/styleguide/shellguide.html*](https://google.github.io/styleguide/shellguide.html)

## **Styling: Spaces vs Tabs**

*Anthony 23/03/2021 comment / debate point : Really doesn't matter but since we're using python, lets follow spaces only with 2 space indentation ?*

* Follow python conventions, indentation **SHOULD** be spaces only with 2 space indentation

## Styling: Shell Naming

### Styling: Shell Variable Naming

|  |  |
| --- | --- |
| **Variables** |  |
| Rules | \* variables **MUST** follow lower\_case\_with\_underscores (also known as *snake\_case*)  \* variables **SHOULD** start with a Noun  \* Variable names **SHOULD** be short yet meaningful.  \* The choice of a variable name **SHOULD** be mnemonic- that is, designed to indicate to the casual observer the intent of its use.  \* One-character variable names **SHOULD** be avoided except for temporary "throwaway" variables. |
| Good Examples | response\_header  commit\_status\_url |
| Poor Examples | run\_thing  header\_statusCode  getStatus |

### Styling: Shell Function Naming

|  |  |
| --- | --- |
| **Functions** |  |
| Rules | \* variables **MUST** follow lower\_case\_with\_underscores (also known as *snake\_case*)  \* variables **SHOULD** start with Verbs |
| Good Examples | get\_response\_header  run\_job |
| Poor Examples | class\_header  runTheJob  Get\_Status |

### Styling: Python Naming Side Rules

|  |  |
| --- | --- |
| **Single character variable names** |  |
| Rules | \* One-character variable names **SHOULD** be avoided except for temporary "throwaway" variables.  \* **MUST** not use the characters 'l' (lowercase letter el), 'O' (uppercase letter oh), or 'I' (uppercase letter eye) as single character variable names.  In some fonts, these characters are indistinguishable from the numerals one and zero. When tempted to use 'l', use 'L' instead. |
| Good Examples | c = 0 # c for temp loop counter :) |
| Poor Examples | I = 0 # see how similar to 1 and l  l = 0 # see how similar to 1, L is better  O = 0 # see how similar to 0 and o |

|  |  |
| --- | --- |
| **Private / Protected Class Attributes** |  |
| Rules | Private variables / function should be prefixed "\_" |
| Good Examples | \_response\_header  \_get\_response\_header |
| Poor Examples | hidden\_variable  \_\_private\_function\_\_ |

|  |  |
| --- | --- |
| **bash array naming** |  |
| Rules | list and dict variables **MAY** include a plural noun  this can help different sequence/maps stdTypes from singular stdTypes aka str/int etc |
| Good Examples | dogs = ("harry", "bonnie") |
| Poor Examples | dog = ("harry", "bonnie") |

|  |  |
| --- | --- |
| **boolean naming** |  |
| Rules | bool variables **MAY** include a possessive verb prefix aka "is\_" "has\_" (also "can" / "any" / "all" / "are") |
| Good Examples | is\_bool: bool = true  has\_variable\_been\_populated = true |
| Poor Examples | get\_header\_already = false  first = true (this one debatably good) |

|  |  |
| --- | --- |
| **Exception / Error naming** |  |
| Rules | Exception **MUST** follow class rules  \* Exception name **SHOULD** end with Error |
| Good Examples | is\_bool = true  has\_variable\_been\_populated = true |
| Poor Examples | get\_header\_already = false  first = True (this one debatably good) |

## Code Documentation

### TODO Comments

* **SHOULD** use TODO/FIXME/WARNING/WORKAROUND inline comments to highlight temporary solutions, a short term workarounds, and other pit falls to incoming developers
* # TODO: need to support oauth2 in near future # FIXME: Code doesn't seems right, this seems to loop infinitely# WORKAROUND: ADO#234 fix for persistent auth bug
* *debatable if useful see* [*https://www.python.org/dev/peps/pep-0350/#objections*](https://www.python.org/dev/peps/pep-0350/#objections) *for lots of counter points*

### Function Comments

* all public functions **SHOULD** be documented using google's format
  + Description of the function.
  + Globals: List of global variables used and modified.
  + Arguments: Arguments taken.
  + Outputs: Output to STDOUT or STDERR.
  + Returns: Returned values other than the default exit status of the last command run.
* ######################################## Cleanup files from the backup directory.# Globals:# BACKUP\_DIR# ORACLE\_SID# Arguments:# None#######################################function cleanup() {…}######################################## Get configuration directory.# Globals:# SOMEDIR# Arguments:# None# Outputs:# Writes location to stdout#######################################function get\_dir() { echo "${SOMEDIR}"}######################################## Delete a file in a sophisticated manner.# Arguments:# File to delete, a path.# Returns:# 0 if thing was deleted, non-zero on error.#######################################function del\_thing() { rm "$1"}
* *see* [*https://google.github.io/styleguide/shellguide.html*](https://google.github.io/styleguide/shellguide.html)

## Script Basics

### Script name

* scripts **SHOULD** be held in a folder at root called "scripts"
* *exception: main executable scripts might be held at root level*
* all scripts **SHOULD** be post fixed with the file extension ".sh"
* *exception: main executable scripts might have no extension just a shebang*

### Security

* Scripts **MUST** be run in account and users with correct permissions
* Scripts **MUST NOT** use SUID or SGID commands, if necessary sudo **SHOULD** be used instead
* aka be very careful with chmod, use sudo instead to get access
* *exception: where the permission of a script is to change permissions, obviously it can edit permissions*

### Command Line Arguments

* All user executable scripts **SHOULD** contain while getopts loop at top of script
* *should be central place for parsing all command line parameters*
* *warning getopts command is black to most see guides for howto use*
* All user executable scripts **SHOULD** contain a -h help command
* *should run a usage function, that explains how to use the script, e.g.*
* ######################################## Prints out application help and usage# Globals:# None# Arguments:# None#######################################usage() { script\_name=`basename "$0"` echo "Description of what application does" echo "" echo -e "\e[1mSyntax:\e[0m" echo " ${script\_name} [-g|h|v|V]" echo "" echo -e "\e[1moptions:\e[0m" echo " -g Print the GPL license notification." echo " -h Print script usage" echo " -v Verbose mode." echo ""}

*see getopts man page* [*https://man7.org/linux/man-pages/man1/getopts.1p.html*](https://man7.org/linux/man-pages/man1/getopts.1p.html)

*see easy to understand tutorial on getopts* [*https://sookocheff.com/post/bash/parsing-bash-script-arguments-with-shopts/*](https://sookocheff.com/post/bash/parsing-bash-script-arguments-with-shopts/)

### Variables Quoting / Usage (there be dragons)

* **SHOULD** use local where possible to reduce scope of variables
* When creating string **SHOULD** use "${var}" over "$var"
* *creates less bugs, more consistent escaping*
* in general **SHOULD** follow rest of google's quoting guidelines
* *Quotes in bash are a complex topic, especially if user/external input is being put into script, need to ensure no jail break or shell injection is possible with malformed input*
* <https://google.github.io/styleguide/shellguide.html#quoting>

## Terminal Output

### Terminal Output: STDOUT STDERR Pipes

* All debug/verbose messages **SHOULD** be hidden by default and exposed via command line -v or -d flags
* Passwords and Credentials **MUST** never be exposed on console output by default
* *exception: occasionally usernames and credentials required for debugging but credentials* ***MUST*** *be obfuscated in this case via regex/sed*
* # Simplistic example for a 64 character apikey, would remove all characters bar the leading/trailing 2 characters for debugging with -d without meaningfully exposing credential obfuscated\_credential = `echo $raw\_credential | sed s/^(.{2})(.+)(.{2})$/$1xxx$3/`
* Non Error console messages **SHOULD** sent to STDOUT (">&1" with is default)
* $ echo "send this to standard console"# With pointless decoration of explicit STDOUT default pipe$ echo "send this to standard console" >&1
* Errors **SHOULD** be sent to the STDERR pipe ">&2"
* $ echo "send this to error" >&2# Google recommend abstracting this behind a Error Functionerr() { echo "[$(date +'%Y-%m-%dT%H:%M:%S%z')]: $\*" >&2}if ! do\_something; then err "Unable to do\_something" exit 1fi

*see echo man details here* [*https://linux.die.net/man/1/echo*](https://linux.die.net/man/1/echo)

*see guide to piping* [*https://thoughtbot.com/blog/input-output-redirection-in-the-shell*](https://thoughtbot.com/blog/input-output-redirection-in-the-shell)

### Terminal Output: Colours

* When using bash terminal colours, **SHOULD** ensure they are used in a clear and obvious way
* echo -e "\e[31mERROR: something message in Red\e[0m"
* When using bash terminal colours, **SHOULD** always reset colour to default "\e[0m"
* # end echo with echo -e "\e[0m"

*see more about colours in bash here* [*https://misc.flogisoft.com/bash/tip\_colors\_and\_formatting*](https://misc.flogisoft.com/bash/tip_colors_and_formatting)

## Versioning

### Software Versioning

* All scripts **MUST** follow semantic versioning standard
* *ps: semantic versioning follows sub set of the PEP-0440*
* Script **SHOULD** expose version via a cli -V or --version flag

#### Semantic versioning Cliffnotes

Given a version number MAJOR.MINOR.PATCH, increment the:

* MAJOR version when you make incompatible API changes,
* MINOR version when you add functionality in a backwards compatible manner, and
* PATCH version when you make backwards compatible bug fixes.

<https://semver.org/>

# Further Reading

* Excellent Google shell best practice
* <https://google.github.io/styleguide/shellguide.html>
* Further reading
* <https://en.wikipedia.org/wiki/Law_of_the_instrument>
* Howto Colours inbash
* <https://misc.flogisoft.com/bash/tip_colors_and_formatting>
* Manual howto Bash
* <https://ss64.com/bash/>
* Nice and Easy tutorial for the new to bash
* <https://ryanstutorials.net/bash-scripting-tutorial/>