Writing Bash Scripts Like A Pro - Part 1 - Styling Guide

6 mins

Published on **25 October 2021**

#bash #programming #tutorial #devops

Writing Bash scripts can be challenging if you don't know the quirks and perks. In my mother tongue, we use the Yiddish word for quirks and perks; it's called "Shtickim" (plural of "Shtick"). Are you ready to learn more about Bash's "Shtickim"?

This blog post is part of a series that I'm working on to preserve the knowledge for future me that forgets stuff, to assist new colleagues, and indulge programmers like you who wish to love Bash as I do. So let's begin, shall we?

It's A Scripting Language

It's important to remember that Bash is a scripting language, which means it doesn't offer the standard functionalities that a programming language has to offer, such as:

- Object-Oriented Programming is not supported natively

- There are no external libraries like Python's requests or Node's axios, though it is possible to use external applications such as curl
- Variables Typing is not supported, and all values are evaluated as *strings*. However, it is possible to use numbers by using specific commands, such as test equality with -eq and increment a variable with ((VAR_NAME+1)). Nevertheless, there's a "weak" way of declaring variables type with the declare command.
- Bash's Associative array like Python's dict or JavaScript's Object is supported from version Bash v4.4, and it's important to remember that macOS is shipped with Bash v3.2 (we'll get to that in future blog posts of this series)
- There is no "source of truth" for naming convention. For example, how would you name a global variable? `Pascal_Case`? `snake_case`? `SCREAMING_SNAKE_CASE`?

As you already guessed, "Bash programmers" (if there is such a thing) face many challenges. The above list is merely the tip of the iceberg.

Here are great blog posts that share the same feelings as I do:

- Foolproof Your Bash Script
- Best Practices for Writing Bash Scripts
- Bash best practices

Now that we've covered the fact that I'm in love with Bash, I want to share that feeling with you; here goes.

Variables Naming Convention

Here's how I name variables in my Bash scripts

Type	Scope	Convention
Environment	Global	MY_VARIABLE
Global	Global	_MY_VARIABLE
Local	Function	my_variable

In my older Bash scripts, the names of the variables were hard to interpret. Changing to this naming convention helped me a lot to understand the scope of variables and their purpose.

Good Vibes Application

And of course, we gotta' see some practical example, so here's how I implement the above naming convention in my `good_vibes.sh` application.

`good_vibes.sh`

```bash
#!/usr/bin/env bash
# ^ This is called a Shebang
# I'll cover it in future blog posts


# Global variables are initialized by Env Vars.
# I'm setting a default value with "${VAR_NAME:-"DEFAULT_VALUE"}"
_USER_NAME="${USER_NAME:-"$USER"}"
_USER_AGE="${USER_AGE:-""}"


complement_name(){
  local name="$1"
  echo "Wow, ${name}, you have a beautiful name!"
}


complement_age(){
  local name="$1"
  local age="$2"
  if [[ "$age" -gt "30" ]]; then
    echo "Seriously ${name}? I thought you were $((age-7))"
  else
    echo "Such a weird age, are you sure it's a number?"
  fi
}


main(){
  # The only function that is not "pure"
  # This function is tightly coupled to the script
  complement_name "$_USER_NAME"
  complement_age "$_USER_NAME" "$_USER_AGE"
}


# Invokes the main function
main
```

good_vibes.sh - Execution and output

```bash
export USER_NAME="Julia" USER_AGE="36" && \
bash good_vibes.sh

# Output
Wow, Julia, you have a beautiful name!
Seriously Julia? I thought you were 29
```

Let's break down the `good_vibes.sh` application to a "set of rules" that can be implemented in your scripts.

Code block spacing

Two (2) blank rows between each block of code make the script more readable.

Indentation

I'm using two (2) spaces, though it's totally fine to use four (4) spaces for indentation. Just make sure you're not mixing between the two.

Curly braces

If it's a `${VARIABLE} concatenated with string`, use curly braces as it makes it easier to read.

In case it's a `"$LONELY_VARIABLE"` there's no need for that, as it will help you realize faster if it's "lonely" or not.

The primary purpose for curly braces is for performing a Shell Parameter Expansion, as demonstrated in the Global variables initialization part.

Squared brackets

Using **double** `[[ ]]` squared brackets makes it easier to read conditional code blocks. However, do note that using double squared brackets is not supported in Shell sh; instead, you should use single brackets `[ ]`.

To demonstrate the readability, here's a "complex" conditional code block:

```
if [[ "$USER_NAME" = "Julia" || "$USER_NAME" = "Willy" ]] \
   && [[ "$USER_AGE" -gt "30" ]]; then
  echo "Easy to read right?"
fi

```

```
# Mind that `||` is replaced with `-o`, see https://acloudguru.com/blog/en
# Thank you William Pursell
if [ "$USER_NAME" = "Julia" -o "$USER_NAME" = "Willy" ] \
   && [ "$USER_AGE" -gt "30" ]; then
  echo "No idea why but I feel lost with single brackets."
fi
```

In case you didn't notice, you've just learned that `||` stands for `OR` and `&&` stands for `AND`. And the short -gt expression means `greater than` when using numbers. Finally, the `\` character allows breaking rows in favor of making the code more readable.

Shtick: Using `\` with an extra space `\ <- extra space` can lead to weird errors. Make sure there are no trailing spaces after `\`.

I assume that using `[[ ]]` feels more intuitive since most conditional commands are doubled `&&` `||`.

Variable initialization

Global variables are initialized with Environment Variables and are set with default values in case of empty Environment variables.

As mentioned in the `good_vibes.sh` comments, I'm setting a default value with

```
"${VAR_NAME:-"DEFAULT_VALUE"}"
```

In the above snippet, the text `DEFAULT_VALUE` is hardcoded, and it's possible to replace it with a variable. For example

```
_USER_NAME="${USER_NAME:-"$USER"}"
```

Functions and local function variables

Functions names and `local` function variables names are `snake_cased`. You might want to change functions names to `lowerCamelCase`, and of course, it's your call.

Coupling a function to the script is a common mistake, though I do sin from time to time, and you'll see Global/Environment variables in my functions, but that happens when I know that "this piece of code won't change a lot".

Oh, and make sure you don't use `$1` or any other argument directly; always use `local var_name="$1"`.

```
_USER_NAME="${USER_NAME:-"$USER"}"

# Bad - coupled
coupled_username(){
  echo "_USER_NAME = ${_USER_NAME}"
}

# Good - decoupled
decoupled_username(){
  local name="$1"
  echo "name = ${name}"
}

# Usage
coupled_username
decoupled_username "$_USER_NAME"
```

Functional Programming

This topic relates to **Functions and local function variables**, where functions are as "pure" as possible. As you can see in `good_vibes.sh`, almost everything is wrapped in a function, except for **Initializing Global variables**.

I don't see the point of writing the `init_vars` function, whose purpose is to deal with Global variables. However, I do find myself adding a `validate_vars` function from time to time, which goes over the Global variables and validates their values. I'm sure there's room for debate here, so feel free to comment with your thoughts.

Final Words

The "Good Vibes Application" mostly covered how to write a readable Bash script following the Functional Programming paradigm.

If you feel that there's a need to change how you name variables and functions, go for it! As long as it's easy to understand your code, you're on the right track.

The next blog posts in this series will cover the following topics:

- Error handling
- Retrieving JSON data from an HTTP endpoint
- Background jobs and watching file for changes with fswatch
- Git Repository structure - adding Bash scripts to existing repositories or creating a new repository with a Bash CLI application
- Publishing a Bash CLI as a Docker image

And more, and more … I'm just going to spit it all out to blog posts. Feel free to comment with questions or suggestions for my next blog posts.

Spread The Word

Facebook Twitter LinkedIn Reddit

Related Posts

- Boosting terminal productivity tips
- MVC design pattern in Terraform
- Parsing command line arguments in Bash
- Fun With Regular Expressions
- How To Develop A Progressive Web Application On An Android Device