CSE 374 Bash Style Guide

A Bash script is a plain text file which contains a series of commands. These commands are a mixture of commands we would normally type ouselves on the command line (such as `ls` or `cp` for example) and commands we could type on the command line but generally wouldn't.

> **Tip**
>
> Anything you can run normally on the command line can be put into a script and it will do exactly the same thing. Similarly, anything you can put into a script can also be run normally on the command line and it will do exactly the same thing.

General

In this class, we will be using Bash to run our executables. Each Bash script must start with `#!/bin/bash` and a minimum number of option flags. Use `set` to set shell options instead, this will prevent your script from breaking when it is executed with `bash <script_name>`.

Bash scripts preferably have no file extension, but may have an `.sh` extension. If you create a Bash script library, that code should always have a `.sh` extension.

> **Note**
>
> This style guide is here to provide a reference to what is recommended. In the end, please use common sense and **BE CONSISTENT**.

Comments

File Headers

Start each file with author information (name, net ID) and assignment information (assignment #). Every file must also have a top-level comment including a brief overview of its contents. For example:

```
#!/bin/bash
#
```

```
# Author: John Doe
# UWNetID: jdoe124
# Homework 1: Part 1
#
# Prints out "Hello, World!" 50 times.
```

Function Comments

It should be possible for someone else to learn how to use your program or to use a function in your library by reading the comments (and self-help, if provided) without reading the code.

If your function is either reasonably long or not obvious on first look, it should have a function comment at the top of it.

All function comments should contain:

- Description of the function
- Global variables used and modified
- Arguments taken
- Returned value other than the exit status of the last command run

Examples:

```
#!/bin/bash
#
# Author: John Doe
# UWNetID: jdoe124
# Homework 99: Part 99
#
# Connect to coffee machine on network and brews a cup
DONE=0

# Function comment is not needed here
# No comment is preferred over "Prints Done! Coffee is ready. to the console"
done() {
  if [[ $DONE -eq 1 ]]; then
    echo -n "Done! "
    echo "Coffee is ready"
  fi
}

# Establishes a connection with the coffee machine, sends a "brew" command,
# and waits for coffee to finish.
#
# Calls done() when coffee is ready.
# Globals:
#   DONE
# Arguments:
#   None
# Returns:
#   None
brew_coffee() {
  ...
}
```

Implementation Comment

If there's a complex algorithm or you're doing something out of the ordinary, put a short comment in. **Don't comment everything.**

Formatting

> **Note**
>
> The formatting styles are suggestions on what new code should look like. In general, you should follow whatever styles that are already there for files that you are modifying. This also means if you pick a formatting style, you should stick with it.

Indentation

Use spaces instead of tabs. You are welcome to use 2 or 4 spaces depending on preference, but make sure to *stay consistent*. When in doubt preference is for 2 spaces.

For readability, you should also separate blocks with blanks lines.

Loops and Conditionals

`; then` and `; do` should be on the same line as the `if/for/while`. `else/elif` should be on their own line and closing statements (`fi/done`) should be on their own line vertically aligned with the opening statement. For example:

```
if <condition1>; then
  ...
elif <condition2>; then
  ...
else
  ...
fi
```

Variable Expansion

In order of precedence: Stay consistent with what you find or have been doing; quote your variables (e.g. `"$var"`). The form `"${var}"` is required in some situations and may be used consistently.

Quoting

In Bash, `'Single Quotes'` indicates that no substitution is desired, e.g., `'*.txt'` will match

with the literal string `*.txt`. `"Double Quotes"` indicates that substitution will happen, e.g., `"*.txt"` will match any string ending with `.txt`.

Naming

Function Naming

Function names should be in `snake_case`. That is, all lower case and words are separated by underscores.

Braces should be on the same line as the function name, and there is no space between the function name and the parentheses.

The `function` keyword is optional, but *be consistent*.

Variables

Variables are generally in `snake_case`, with underscores separating each word; constants and anything exported to the environment should be `ALL_CAPS`.

Some Other Style Recommendations

`test, [, [[`

In general, `[[` is preferred over `test` or `[` because it reduces errors as no pathname expansion or word splitting takes place in between `[[ ... ]]` and it allows for regex matching with string.

Testing Strings

When testing to see if a string is empty, explicitly use `-n` (string is not length zero) and `-z` (string is length zero). For example:

```
# -z (string length is zero) and -n (string length is not zero) are
# preferred over testing for an empty string
if [[ -z "${my_var}" ]]; then
  ...
fi

# This is OK (ensure quotes on the empty side), but not preferred:
if [[ "${my_var}" = "" ]]; then
  ...
fi
```

If you find any bugs in this tutorial, please e-mail them to the CSE 374 staff.

Author: Andrew Tran (atran35@cs)

Edited by: Cynthia Zhang, Megan Hazen, Soumya Vashist

Adapted from the Google Shell Style Guide.