5 Mistakes To Avoid For Writing High-Quality Bash Comments

Every programming language, including **Bash**, support a form of **comments**. As one of the key properties of a good shell script is to be reusable, it implies to be human-readable for anyone using the script, including yourself months later. Properly using *bash comments* ensure that your script can be maintained easily. Comments in Bash scripts are often overlooked and underappreciated, leading to hard to debug problems and hours of frustrations.

How to Comment Lines in a Bash Script?

Bash comments can only be done as *single-line comment* using the hash character `#`. Every line or word starting by the `#` sign cause all the following content to be ignored by the bash shell. This is the only way to do a bash comment and ensure text or code is absolutely *not evaluated* in Bash.

The only exception is the first line of a script with the shebang `#!` which is used to specify the script interpreter to be used on Linux and Unix platforms. This is still considered a comment in Bash and ignored by the Bash interpreter.

For multiple lines comments or block comments in Bash, see the Mistake #5 Using Unconventional Block Comments.

```
#!/bin/bash
# This is a single-line comment in bash

echo "Hello World!" # This is an inline comment in bash
```

⚠ In an interactive shell, when the option `interactive_comments` is disabled, the comments won't be allowed. If you try to use comments in such a scenario you will get the following error `bash: #: command not found`. You can fix this by using `shopt -s interactive_comments`.

The 5 Mistakes To Avoid

1. Not Having a File Header Comment

This is part of programming 101 and it is also true when writing a script in Bash. All scripts must start with a comment that describes the purpose of the script. This is often called a *File Header*. Not having a file header prevent users to quickly understand and utilize your script without reading the full script. In some cases, it will also be the place where you will want to provide the authors' names and include a copyright notice.

> Note, that including a proper copyright notice in each source file is a requirement under many open source licenses, including the GPL.

```bash
#!/bin/bash
# This is an example of a File Header comments in Bash
#
# Copyright 2020 John Doe
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>.

echo "Hello World!" # This is an inline comment in bash
```

2. Missing Function Comments

Commenting functions is also a recommended programming practice. Generally, all bash functions should be commented. An exception can be made for small and obvious functions in a simple script. Libraries should always have their functions commented. Function comments in bash should state the following as necessary:

- Description of the function
- List of global variables used or modified
- Arguments taken by the function
- Outputs of the function to STDOUT and STDERR
- Returned values other than the exit status of the last command run

```bash
#######################################
# Print a given string
# GLOBALS:
#   A_STRING_PREFIX
# ARGUMENTS:
#   String to print
# OUTPUTS:
#   Write String to stdout
# RETURN:
#   0 if print succeeds, non-zero on error.
#######################################
function print_string() {
  echo "${A_STRING_PREFIX}: $1"
}
```

3. Making Long and Verbose Comments

You should comment on all the important and complex parts of your code, including non-obvious or know-how sections. Though, this doesn't mean you should comment on everything. A simple echo line or a bash arithmetic addition is not warranted a long and verbose comment.

Don't make your bash script longer and harder to work with by using overly long comments.

4. Not Using Consistent Labeling for TODO Comments

Every program, including your bash script, that is a work in progress should be open to future modification and improvement. When doing your first few iterations, you may find yourself adding comments for some `todo` tasks. It is important to stay consistent on how you label comments across your source codebase. This allows you to easily find and list all your *TODO* tasks within your git repository using a command line like `git grep -I -l TODO | xargs -n1 git blame | grep TODO`.

Your label can be whatever you want, *TODO* is the most obvious one being used. The important point is consistency across your code repository. Depending on your use case, you may also want to include the name and email address of the person making the comment who may be the subject matter expert to resolve or discuss this *TODO* task.

```
# TODO(John Doe, john.doe@example.com): This is an example of TODO comment in bash (bug ####)
```

5. Using Unconventional Block Comments

Let's be clear, **there is no block comment in Bash**. *Block comments* are often called *multiple lines comments*. Any alternative in Bash that mimics block comments is a convenience that can be error-prone. A common alternative is to use the *heredoc* notation with the bash null command. This is acceptable to do in some edge cases when debugging, troubleshooting, or while refactoring a large portion of code. Though, this shouldn't be a standard way to comment in Bash.

One practical reason to stick to the hash mark `#` to comment in bash is that it allows a user to quickly `grep` through a script and get all the human-readable comments to understand the shell script.

If you want multiline comments in bash, and choose to use the heredoc notation with the bash null command, then ***make sure to single quote the heredoc delimiter otherwise variables will be expanded***.

```
[me@linux ~]$ : <<'COMMENT'
This is a multiline block comment in bash
using the heredoc delimiter with single quotes
COMMENT
```

The Complete How To Guide of Bash Functions

How To Create Simple Menu with the Shell Select Loop?

What is the Right Way to do Bash Loops?

How To Script Error Free Bash If Statement?

How To Format Date and Time in Linux, macOS, and Bash?

How To Do Advanced Math Calculation Using bc?