**How to Create a Foolproof Interactive Terminal Menu With Bash Scripts**

**Color-coordinated multiple-choice menu for any action-oriented terminal app**

Text

Description automatically generated

Interactive terminal menu. Image by the author.

**Introduction**

If you are thinking of creating an interactive menu for your next Bash Script project, you are at the right place. In this article, we will create a simple menu template with colors for ease of navigation.

Terminal commands use options to pass parameters to a program. These options can be a dash followed by one letter or two dashes with a word. If the option takes an argument it is called a switch, or otherwise, it is called a flag.

Using a menu system for the terminal command makes it easy to navigate if your app has multilayers of commands. I used this method to create a [terminal radio](https://levelup.gitconnected.com/discover-rejuvenate-your-music-library-with-command-line-f3268db67bba) with colors added.

**Main menu**

The main menu function holds two menu items but you can add more for your project.

[See the code in action.](https://tech.io/snippet/lfFEw6c)

Text

Description automatically generated with low confidence

The echo command with-n option keeps the cursor in the same line Image by the author.

The echo -e option allows us to use backslashes to escape characters, for example, a new line \n or a tab \t, if you need them. The echo -n option omits the trailing newline. Without the -n option, the cursor goes to the next line as you see in this image.

A picture containing text

Description automatically generated

The echo command without-n option append a newline. Image by the author.

All menus will contain the case statement to hold different choices. It makes it readable and more manageable than using multiple if statements.

If a user select 1, it will show the sub-menu. You can change CMD1 to any menu name. Option 0 will exit the script and any other letters, \*, will show a warning, and exit with the status of 1.

**Sub-menu**

The first sub-menu item, SUBCMD1, holds a subcommand that shows a sub-sub-menu. Once the sub-sub-menu completes the action, it will show the submenu again. The second menu item, Go Back to Main Menu sends you back to the main menu using the menu function in the case statement.

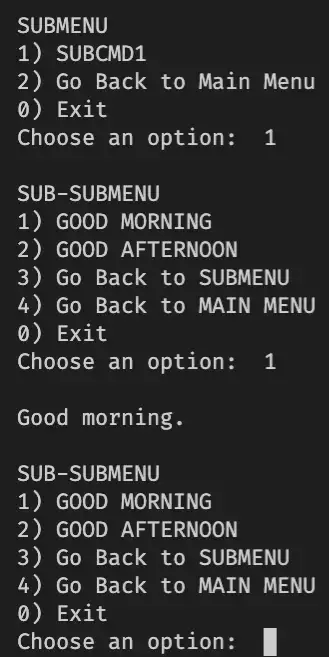
Text

Description automatically generated

The submenu includes Go Back to Main Menu. Image by the author.

**Sub-sub-menu**

We add simple functions for the first and second items in the sub-sub-menu. The third and fourth items will return to the submenu and main menu. We are repeating the same for 0 and \* so let’s create common functions in the next section.



Sub-submenu. Image by the author.

**Creating common functions**

Let’s create functions for the Exit and Wrong option:

Then we can apply them to 0 and \* in our case statements.

case $ans in

...

0)

fn\_bye

;;

\*)

fn\_fail

;;

esac

...

So far the menu is plain. Let’s add colors to our menus in the next section.

**Creating color functions**

These color functions can take an argument. For example:

$(greenprint '1)') SUBCMD1

$(magentaprint '2)') Go Back to Main Menu

$(redprint '0)') Exit

We use $(command) inside the quotes. It runs the command and replaces its output.

For our menu, we are going to use magenta for Go Back to Main Menu and red for Exit, etc to make it consistent throughout the menu. Also, we will add different colors to menu names.

$(magentaprint 'MAIN MENU')

$(blueprint 'CMD1 SUBMENU')

$(yellowprint 'SUB-SUBMENU')

**Final code**

Let’s put it all together. In the final section of this article, you can test the script with an embedded interactive terminal console. Please have a go at navigating the different menu points.

Text

Description automatically generated

Menus using with color functions. Image by the author.

[See the final code in action.](https://tech.io/snippet/08FHqZR) Image by the author.

The following is the final code in action.

**Conclusion**

I used this method in my terminal radio called tera. Please[install it](https://github.com/shinokada/tera) and see how I applied these menu systems in a real-life application.

You can be creative and use emojis or Figlet letters for the menu items as well.

An interactive menu system makes it easier to navigate through a complex menu system than a normal option parser system if your application has multilayered menu systems.

I hope you can apply this code to your next Bash script projects.