Scripting for Kiddies

Getting into Arguments with a Script

Irfan Kanat

2023-05-26

Options to A Script

The arguments already made our script much more powerful. The problem is these arguments are positional. That means if you were to change the order of input, you could mess things up big time.

If you are familiar with unix commands you will note that options (-h, -v, etc.) alter the running of these commands significantly.

We can achieve the same with our scripts. The parameters that start with a dash will be treated as options.

Just as an exercise let us add help functionality to our prior address script. We need to use getopts. The syntax is a bit gnarly. Here is what you need to pay attention to:

First you provide a list of options to look for separated by columns: while **getopts ":h" option;**

Then provide cases for each option: case **$option in; h)**

Note the use of double semicolons to indicate the end of each case: exit**;;**

Let us define one option for help functionality.

```
#! /bin/bash

# Declare Help as a function and keep it tidy
Help (){
    # DISPLAY HELP
    echo "Enter a list of files separated by spaces."
    echo "The script will print out file sizes."
    echo "Use -h to display this help prompt"
}

# Get Options
while getopts ":h" option;
do
    case $option in
        h)
            Help
            exit;;
    esac
done
```

```
echo "Welcome to $0 script"
echo "There are $# arguments."
# Iterate through arguments and get size for each
for file in "$@"
    do
        echo "$(du -sh $file)"
    done
```

Save the script as size4.sh and make it executable.

Now test it with the -h option.

```
./size4.sh -h
```

Same script with prior information.

```
./size4.sh /etc/passwd Desktop/*
```

Handling Invalid Options

The script will keep working unless we specify a case with unknown arguments.

```
./size4.sh -q /etc/passwd Desktop/*
```

We likely would like the script to error out. We want to add a case for the unknown parameters. We exit the script with exit code 2 which indicates error.

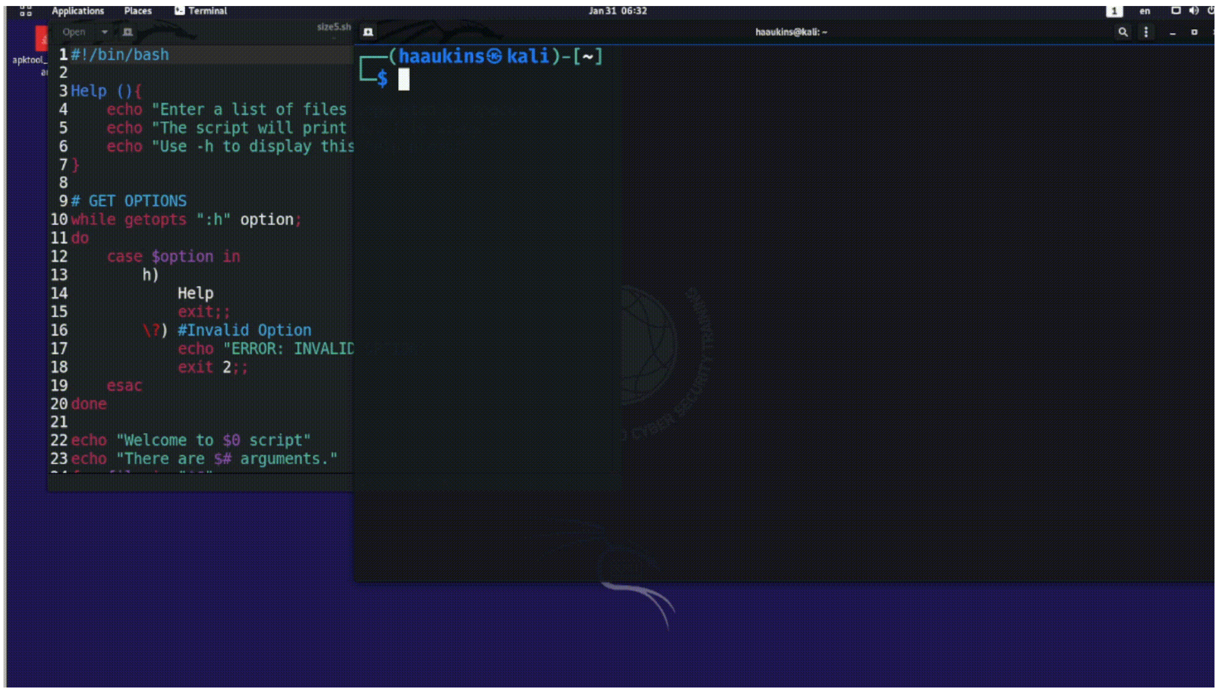
```
#! /bin/bash

Help (){
    # DISPLAY HELP
    echo "Enter a list of files separated by spaces."
    echo "The script will print out file sizes."
    echo "Use -h to display this help prompt"
}

# Get Options
```

```
while getopts ":h" option;
do
    case $option in
        h)
            Help
            exit;;
        \?) # INVALID OPTIONS
            echo "Error: Invalid option"
            exit 2;;
    esac
done


echo "Welcome to $0 script"
echo "There are $# arguments."
# Iterate through arguments and get size for each
for file in "$@"
    do
        echo "$(du -sh $file)"
    done
```

Test it by running with invalid options.

```
./size5.sh -qwerty /etc/passwd Desktop/*
```

Also do test that it runs correctly as before.



```
#!/bin/bash

Help (){
    echo "Enter a list of files
    echo "The script will print
    echo "Use -h to display this
}

# GET OPTIONS
while getopts ":h" option;
do
    case $option in
        h)
            Help
            exit;;
        \?) #Invalid Option
            echo "ERROR: INVALID
            exit 2;;
    esac
done

echo "Welcome to $0 script"
echo "There are $# arguments."
```

```
┌──(haaukins㉿kali)-[~]
└─$ 
```

Using Options to Get Around Ordering Issues

When you have multiple arguments that all need to be treated the same way $@ works just fine. What if you have arguments that require different treatment? One way is to specify a strict order to arguments. First argument is always the input, last argument is always the output… As you can guess, this gets old quick. A more elegant solution is to use options.

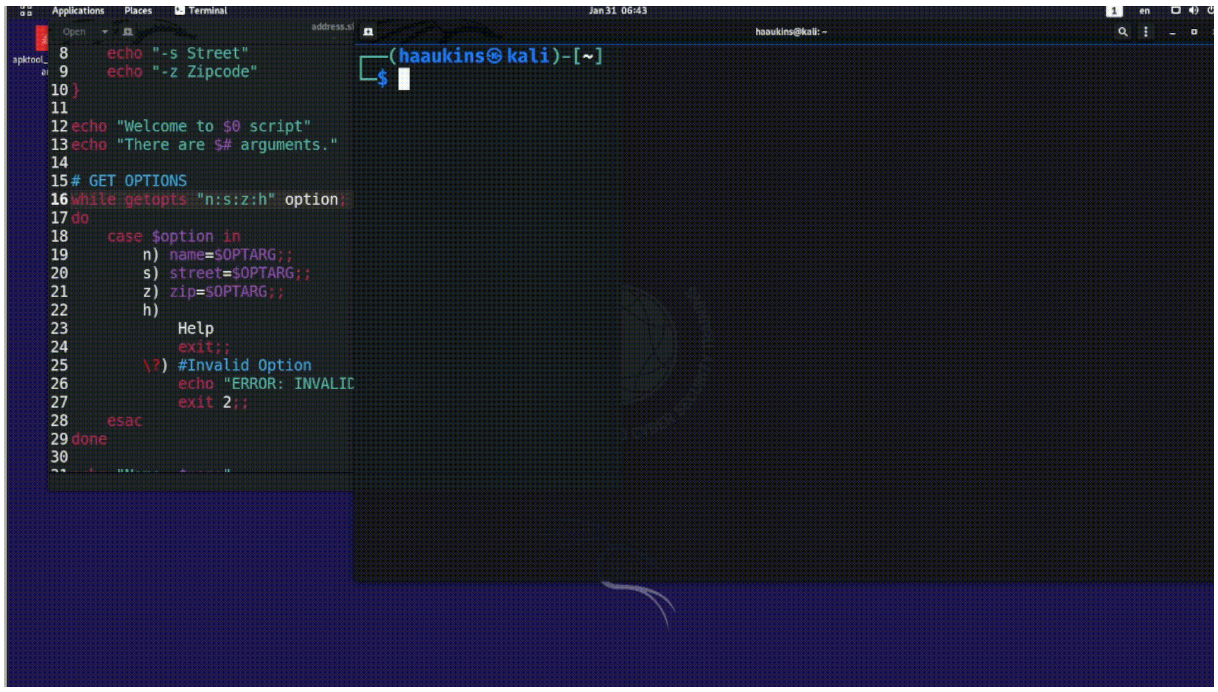
Below is a simple example of an address script.

```
#! /bin/bash

Help (){
    # DISPLAY HELP
    echo "The script formats address information."
    echo "Use -h to display this help prompt."
    echo "Use the following options to declare parameters to the script."
    echo "-n Name"
    echo "-s Street"
    echo "-z Zip Code"
}
```

```
# Get Options
while getopts "n:s:z:h" option;
do
    case $option in
        n)  # Set Name
            name=$OPTARG;;
        s)  street=$OPTARG;;
        z)  zip=$OPTARG;;
        h)  # Display Help
            Help
            exit;;
        \?) # INVALID OPTIONS
            echo "Error: Invalid option"
            exit 2;;
    esac
done


echo "Welcome to $0 script"
echo "Address information as follows"
echo "Name: $name"
echo "Street: $street"
echo "Zip: $zip"
```

You can see the usefulness of this approach. Specify input and output files, alter state of execution (backup, restore, ransom)…

Below you can see how options help us avoid issues with ordering.



```
    echo "-s Street"
    echo "-z Zipcode"
}

echo "Welcome to $0 script"
echo "There are $# arguments."

# GET OPTIONS
while getopts "n:s:z:h" option;
do
    case $option in
        n) name=$OPTARG;;
        s) street=$OPTARG;;
        z) zip=$OPTARG;;
        h)
            Help
            exit;;
        \?) #Invalid Option
            echo "ERROR: INVALID
            exit 2;;
    esac
done
```

```
┌──(haaukins㉿kali)-[~]
└─$ 
```