BASH Executing Commands

...

TLO KNOWLEDGE AND SKILLS

Conditions:

- Given a classroom, applicable references, and a practical exercise, the Cyber Mission Force student will demonstrate an understanding of Bash executing commands..

Knowledge:

- Identify bash CLI.
- Understand the structure of bash scripts and their structure.

Skills:

- Working knowledge of how to create a basic bash script.

OBJECTIVES

- Analyze, create, and compile simple Bash Scripts
- Use the CLI to execute
- Describe scripting structures

ARRAY

An array is simply a variable that can contain multiple values.

Just like creating a simple variable we will set our values equal to whatever we want to name our array.

For normal arrays you must include parentheses () and separate your values with a space delimiter.

```
myarray=(1 3 2 "cat" "dog" 6 "five" 6)
```

ARRAY

To access our array values we use the $ and the variable name. If we were to only do this our array would output the first value of our array and not our entire array.

In order to access all or specific values of our array we must use curly braces along with brackets and the index-value we want to see.

```
myarray=(1 3 2 "cat" "dog" 6 "five" 6)


echo ${myarray[@]}

echo ${myarray[3]}
```

ARRAY EXAMPLE

```
---

#!/bin/bash


read -p "What number would you like to convert? " decimal

DecimalToBinary=({0..1}{0..1}{0..1}{0..1}{0..1}{0..1}{0..1}{0..1})

echo "$decimal in binary is ${DecimalToBinary[$decimal]}"

---
```

ASSOCIATIVE ARRAY

An associative array allows us to create an array that contains key, value pairs.

By asking for the key we can return a value, and by asking for an index number we can return a pair.

To create this type of array you must use declare with the -A option for associative array, then you will place your key in brackets set equal to your value.

```
declare -A phonebook=( [Bob]=”123-456-7890” [Mary]=”098-765-4321” [Sue]=”567-098-4321” )

echo ${phonebook[@]}

echo ${!phonebook[@]}

echo ${phonebook[Mary]}
```

ASSOCIATIVE ARRAY

To add new elements you set the array with the key equal to the value.

```
phonebook[John]="321-890-7654"

echo ${phonebook[@]}

echo ${!phonebook[@]}
```

SPECIAL VARIABLES

$? This holds the exit value of the last executed command.

If a command exits successfully this status will be a 0 and if it exits with a failure the exit value will be 1.

```
#!/bin/bash


cat /etc/passwd >> baseline.txt
if [[ $? -ne 0 ]]
  then
    echo "Error. Check your permissions and try again."
fi
```

SPECIAL VARIABLES

$! This contains the process ID of the most recent executed command in the background.

---

nano&

echo $!

SPECIAL VARIABLES

$$ This hold the PID of the BASH shell

SPECIAL VARIABLES

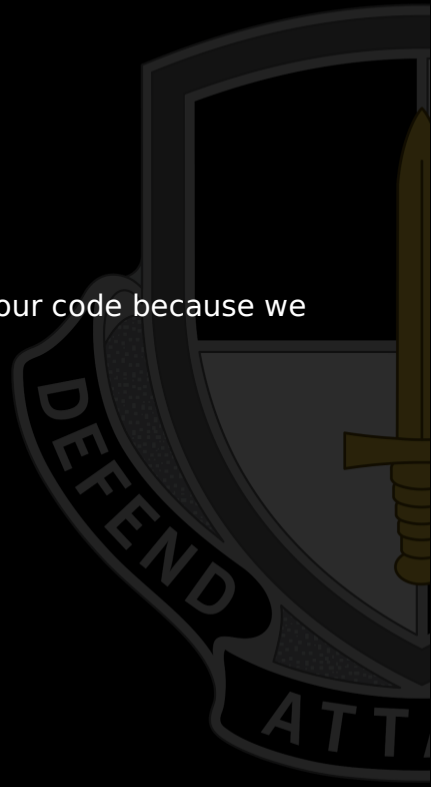
$_ This holds the next item in the pipeline

SUBSHELLS

Subshells allow you to open a new shell and execute the specified commands.

To call a subshell you simply put your commands within parentheses.

When we do this any variables within the subshell will not be accessible by the remainder of your code because we have opened a new shell.



SUBSHELLS

```
----

#!/bin/bash

echo -e "$(pwd) is my current location\n"


(cd ~/Documents

a="This is my variable created and called in my subshell"

echo -e "$a\n"

echo -e "$(pwd) is my current location in my subshell\n")


echo -e "$(pwd) is my current location after running my subshell\n"

echo -e "$a and called outside my subshell"

----
```

SUBSHELLS

1. What's the difference between a variable and an array?

   An array hold multiple values while a variable only holds one.

2. What is the syntax to call a subshell within a script?

   ()

3. What does $? stand for in BASH?

   This is the variable for the exit value of the last executed command.

PATH

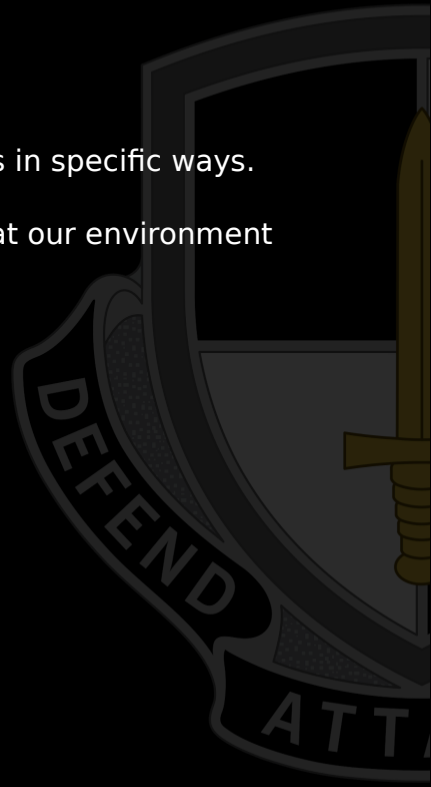
ADDING LOCATION TO PATH DEMO

SET COMMAND

The set command allows us to modify our BASH environment to run our scripts and commands in specific ways.

This can be useful when scripting to not only help up troubleshoot our scripts but to ensure that our environment matches what is required for our script to execute successfully.

To use the set command we use the - to set options whereas to unset them we use the +.



SET COMMAND

-x debugging

This option allows us to see our code execute line by line which can help us to debug code that may not be functioning as intended.

```
#!/bin/bash

set -x

for i in {0..10}

do

  echo $i

  sleep 1

done
```

SET COMMAND

-a export variables

This option allows us to export variables by default so we don't have to run the builtin export command for every script that we write.

```
my_var="This is my variable defined in my terminal"
```

Now write and run the following script

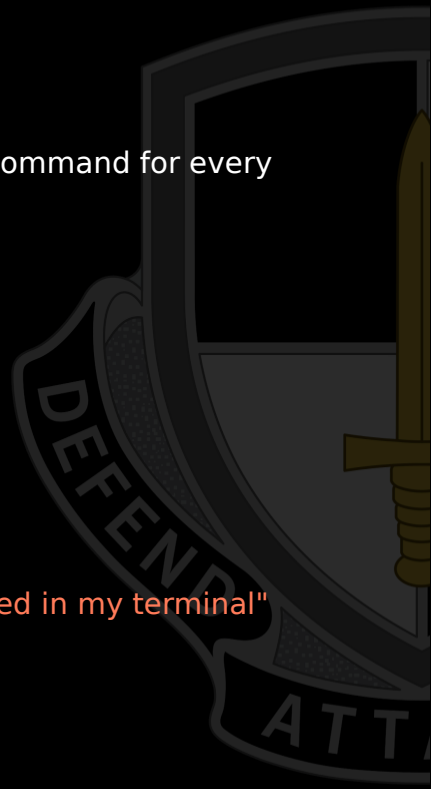
```
#!/bin/bash

echo "$my_var and run in my script"
```

```
set -a

my_var="This is my variable defined in my terminal"

./ourscript.sh
```



SET COMMAND

-u unset variables

This option will output an error when we attempt to run a script with an unset variable. This ensures that our scripts are written efficiently and that we aren't using variables before we assign them.

```
#!/bin/bash


a="This is my only variable"


echo $a $b
```

Now run it with set -u

SET COMMAND

-e stop on non-zero value

This option will stop our script if a non-zero value has been returned. This means that if there's an error that occurs our script will stop.

```
#!/bin/bash

set -e

mkdir ~/test

echo "Mkdir Successful."

touch ~/test/file

echo "Created file ~/test/file"

cat ~/file

echo "Successfully cat ~/file"
```

SET COMMAND

1. What does set -x do?

   debugging

2. How do you unset environmental settings using the set command?

   set +