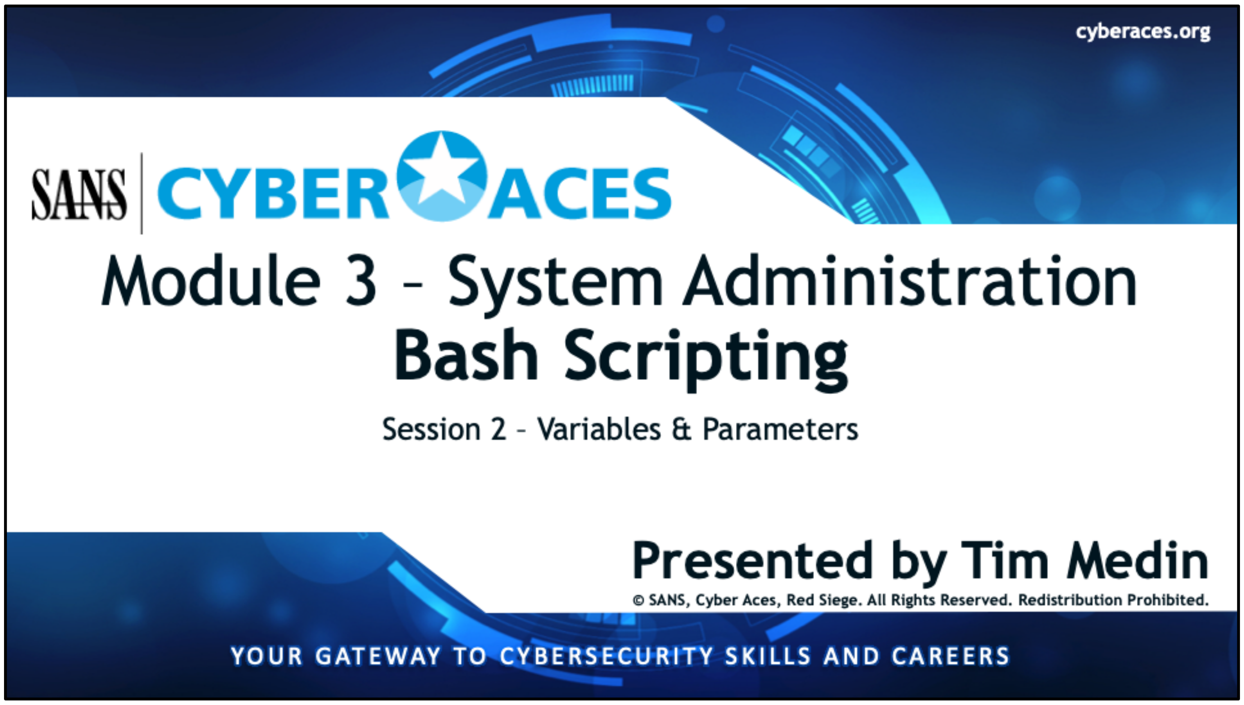


cyberaces.org

SANS | CYBER ACES

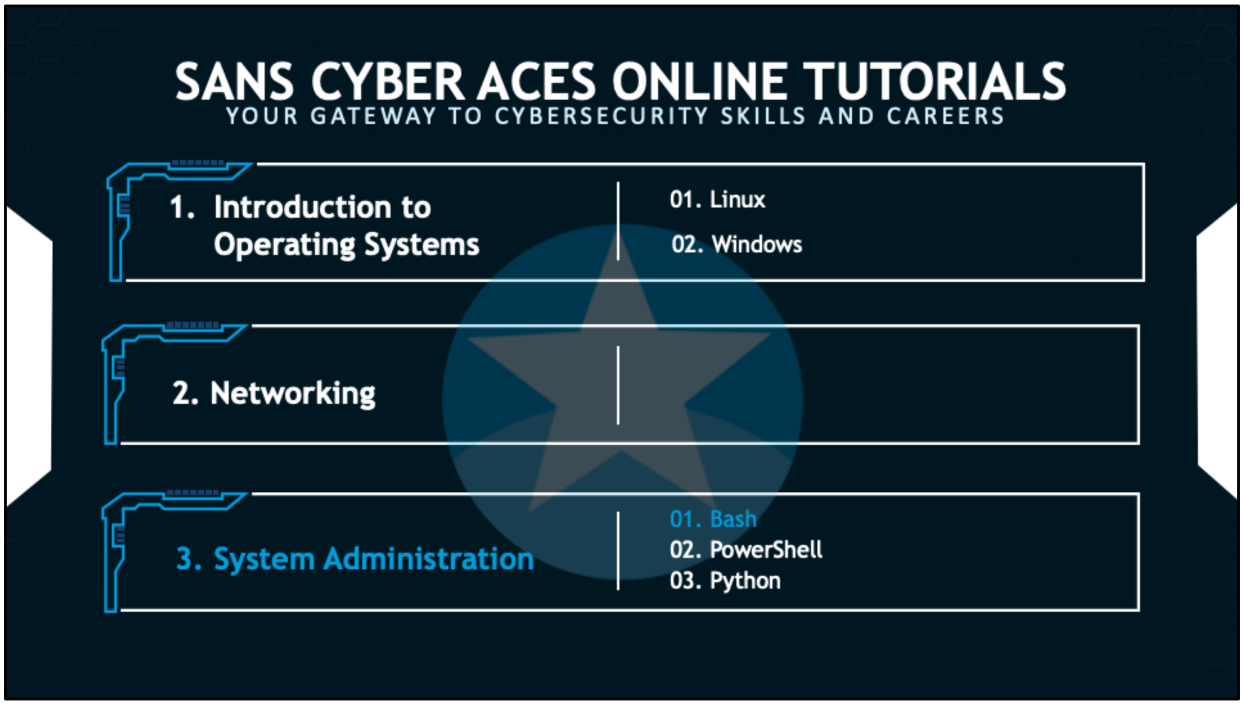
Module 3 – System Administration Bash Scripting

Session 2 – Variables & Parameters

Presented by Tim Medin

YOUR GATEWAY TO CYBERSECURITY SKILLS AND CAREERS

Welcome to Cyber Aces, Module 3! This module provides an introduction to the Bash Scripting. The discussion in this session will be about variables and parameters.



SANS CYBER ACES ONLINE TUTORIALS

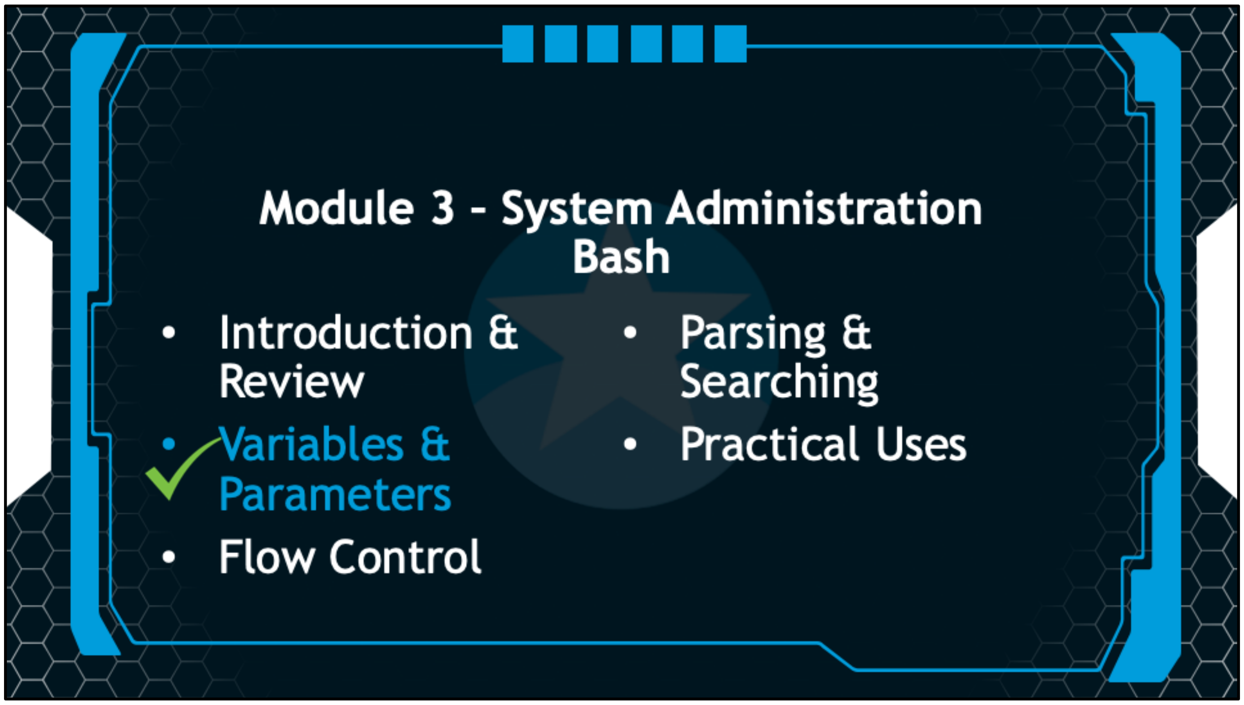
YOUR GATEWAY TO CYBERSECURITY SKILLS AND CAREERS

1. Introduction to Operating Systems	01. Linux 02. Windows
2. Networking	
3. System Administration	01. Bash 02. PowerShell 03. Python

This training material was originally developed to help students, teachers, and mentors prepare for the Cyber Aces Online Competition. This module focuses on the basics of system administration and scripting. . This session is part of Module 3, System Administration. This module is split into three sections, Bash, PowerShell, and Python. In this session, we will continue our examination of Bash.

The three modules of Cyber Aces Online are Operating Systems, Networking, and System Administration.

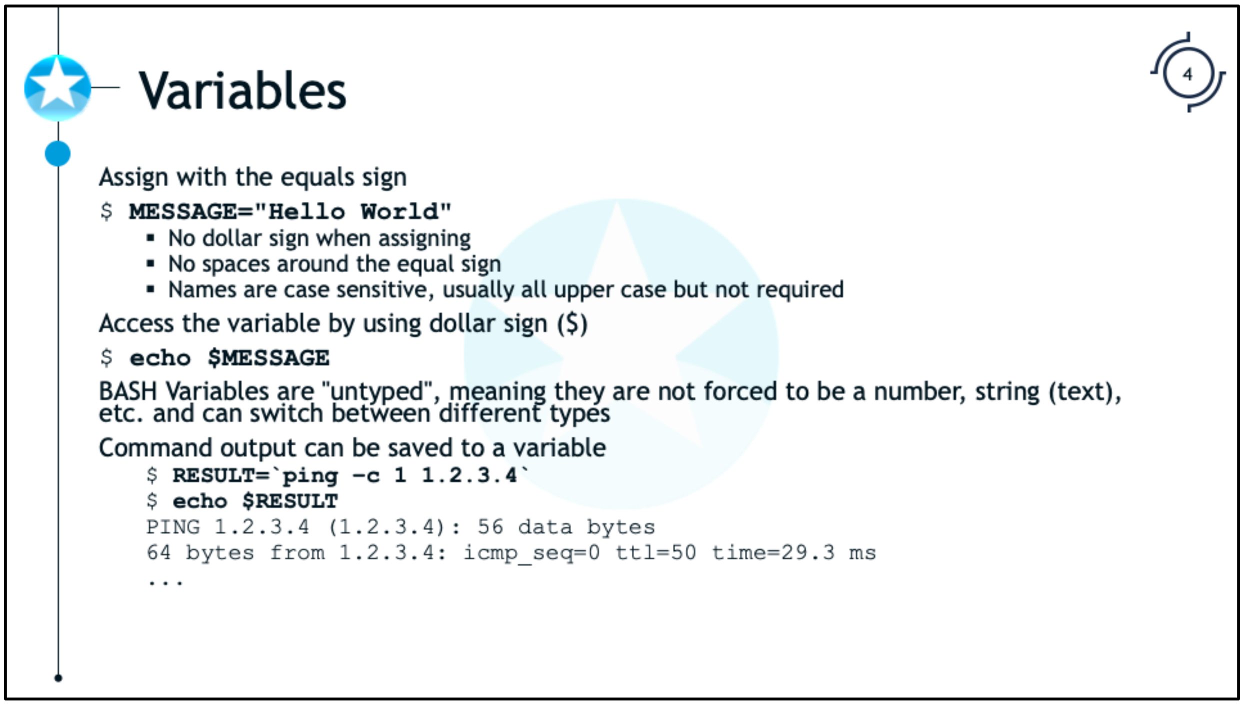
For more information about the Cyber Aces program, please visit the Cyber Aces website at https://CyberAces.org/.



Module 3 - System Administration Bash

- Introduction & Review
- ✓ Variables & Parameters
- Flow Control
- Parsing & Searching
- Practical Uses

In this section, you will be introduced to variables, syntax, and script parameters.



Variables

Assign with the equals sign

```
$ MESSAGE="Hello World"
```

- No dollar sign when assigning
- No spaces around the equal sign
- Names are case sensitive, usually all upper case but not required

Access the variable by using dollar sign ($)

```
$ echo $MESSAGE
```

BASH Variables are "untyped", meaning they are not forced to be a number, string (text), etc. and can switch between different types

Command output can be saved to a variable

```
$ RESULT=`ping -c 1 1.2.3.4`
$ echo $RESULT
PING 1.2.3.4 (1.2.3.4): 56 data bytes
64 bytes from 1.2.3.4: icmp_seq=0 ttl=50 time=29.3 ms
...
```

Variables are representations of memory locations in which we can store arbitrary values. For example, at a Bash prompt we can create a variable called "MESSAGE" and assign it the value "Hello World" by typing the following:

```
$ MESSAGE="Hello World"
```

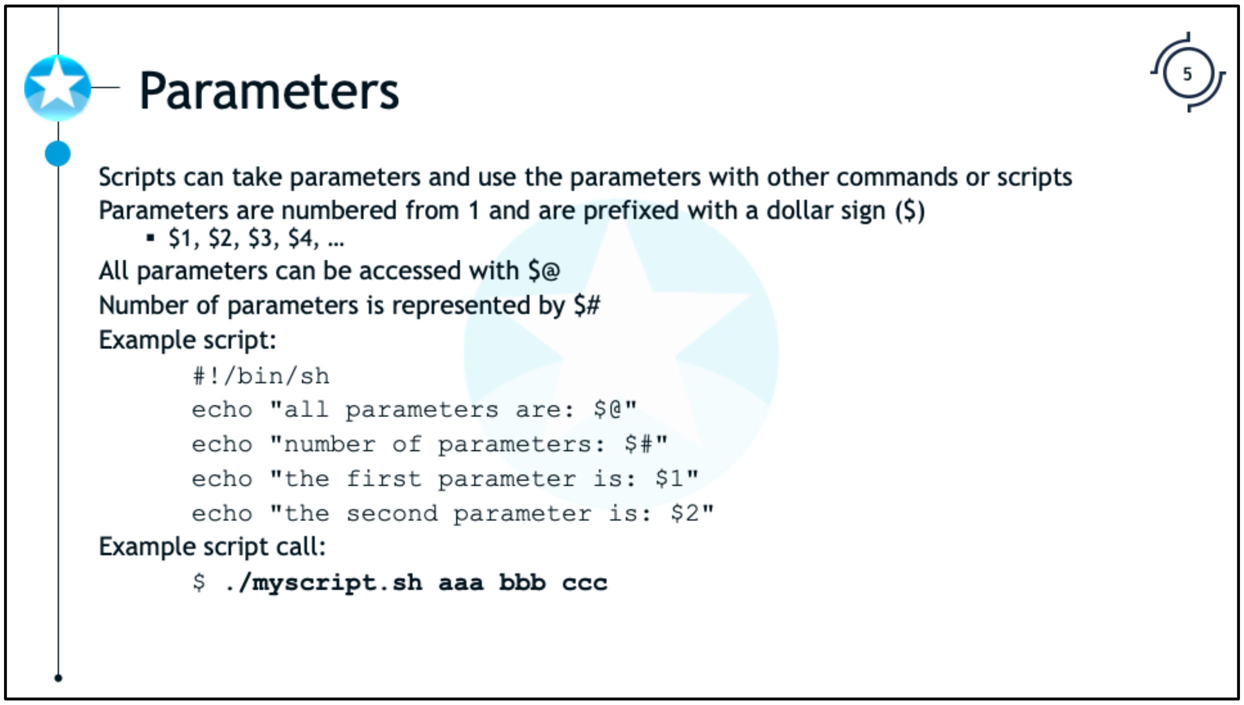
We can then retrieve the contents of that variable with the "echo" command:

```
$ echo $MESSAGE
```

Note that when assigning a variable, you do not prefix its name with a dollar sign, but when accessing it you do. Also note that there is no whitespace between the variable name, the equals sign, and the value.

We can also store the results of the executed programs in variables by using inline process execution (aka Command Substitution). For example:

```
$ RESULT=`ping -c 1 192.168.100.1`
$ echo $RESULT
PING 192.168.100.1 (192.168.100.1): 56 data bytes
64 bytes from 192.168.100.1: icmp_seq=0 ttl=64
...trimmed for brevity...
```



Parameters

Scripts can take parameters and use the parameters with other commands or scripts
Parameters are numbered from 1 and are prefixed with a dollar sign ($)
- $1, $2, $3, $4, …

All parameters can be accessed with $@
Number of parameters is represented by $#
Example script:

```
#!/bin/sh
echo "all parameters are: $@"
echo "number of parameters: $#"
echo "the first parameter is: $1"
echo "the second parameter is: $2"
```

Example script call:

```
$ ./myscript.sh aaa bbb ccc
```

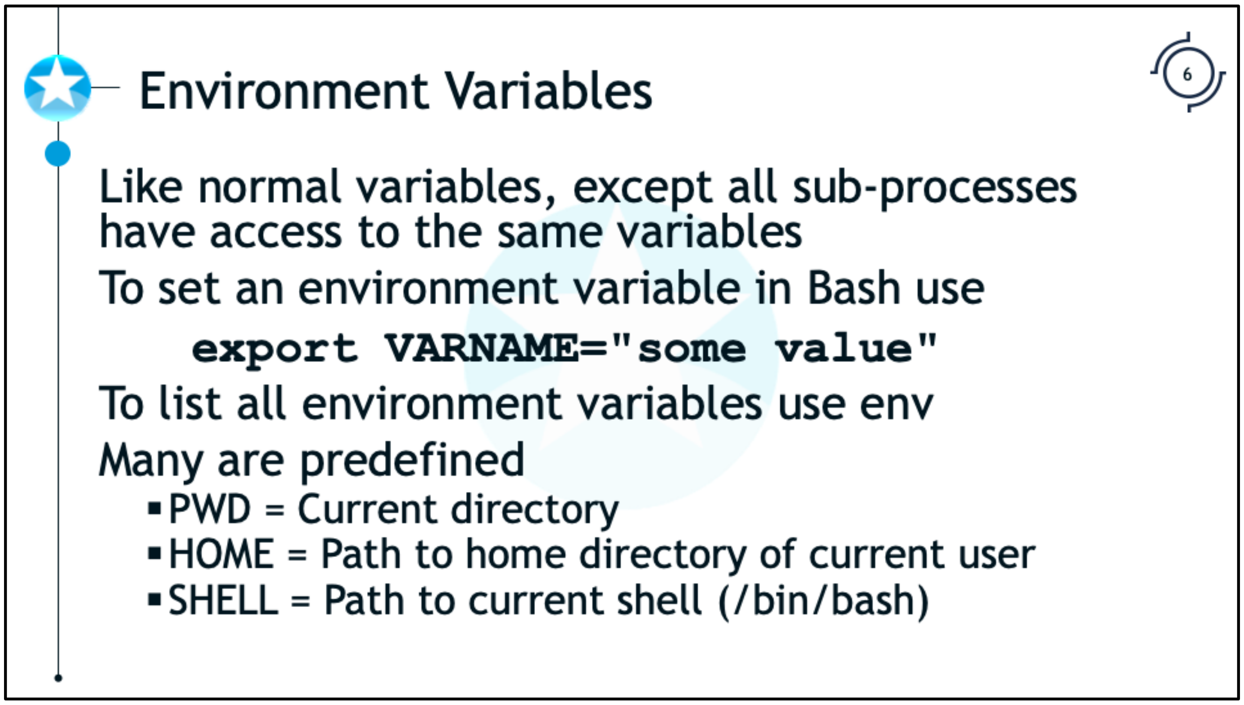
As an alternative to inline commands, you could place your commands inside of another Bash script and pass the output of one command to your new script. Parameters in a BASH script are referred to by their position on the command line. "$1" is the first parameter, "$2" is the second parameter, etc. "$@" is all of the parameters that were passed on the command line.

The contents of our sample script, test.sh:

```
#!/bin/sh
echo "all parameters are: $@"
echo "number of parameters: $#"
echo "the first parameter is: $1"
echo "the second parameter is: $2"
```

A sample execution of our script:

```
$ sh test.sh aaa bbb ccc
all parameters are: aaa bbb ccc
number of parameters: 3
the first parameter is: aaa
the second parameter is: bbb
```



Environment Variables

Like normal variables, except all sub-processes have access to the same variables

To set an environment variable in Bash use

```
export VARNAME="some value"
```

To list all environment variables use env

Many are predefined

- PWD = Current directory
- HOME = Path to home directory of current user
- SHELL = Path to current shell (/bin/bash)

Environment variables are variables used by the system and applications to read configuration settings from the system. The environment variables communicate information such as the user's home directory, the current shell, the username, and many other settings. This makes it easier for applications to interact with the system in a consistent fashion.

To set an environment variable we use the command "export". This environment variable will be set for all processes launched from the parent process. I.E. all processes launched from the shell where the "export" command is used.

Some environment variables are predefined.

PWD: The current directory

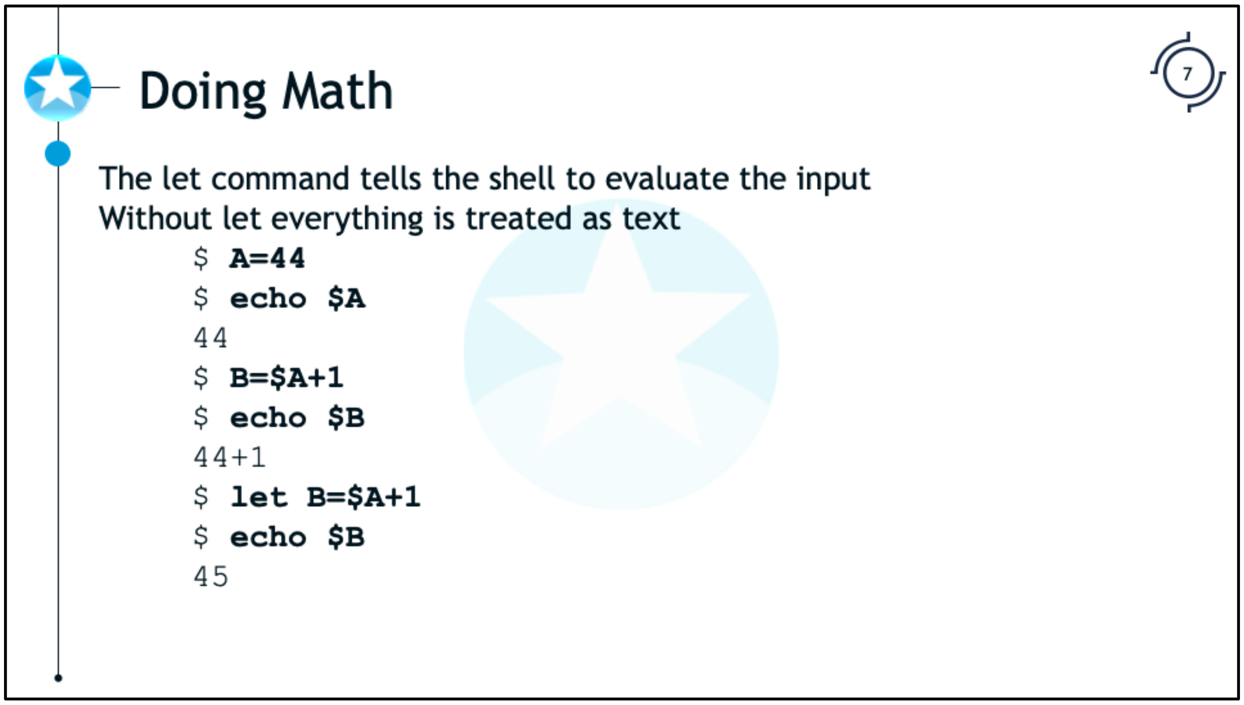
HOME: The path to the user's home directory

SHELL: The path to the current shell (e.g. /bin/bash)

PATH: The directories the shell will search to look for executables

USER & USERNAME: The name of the current user

You can look at all the environment variables set in your shell by typing "env".



Doing Math

The let command tells the shell to evaluate the input
Without let everything is treated as text

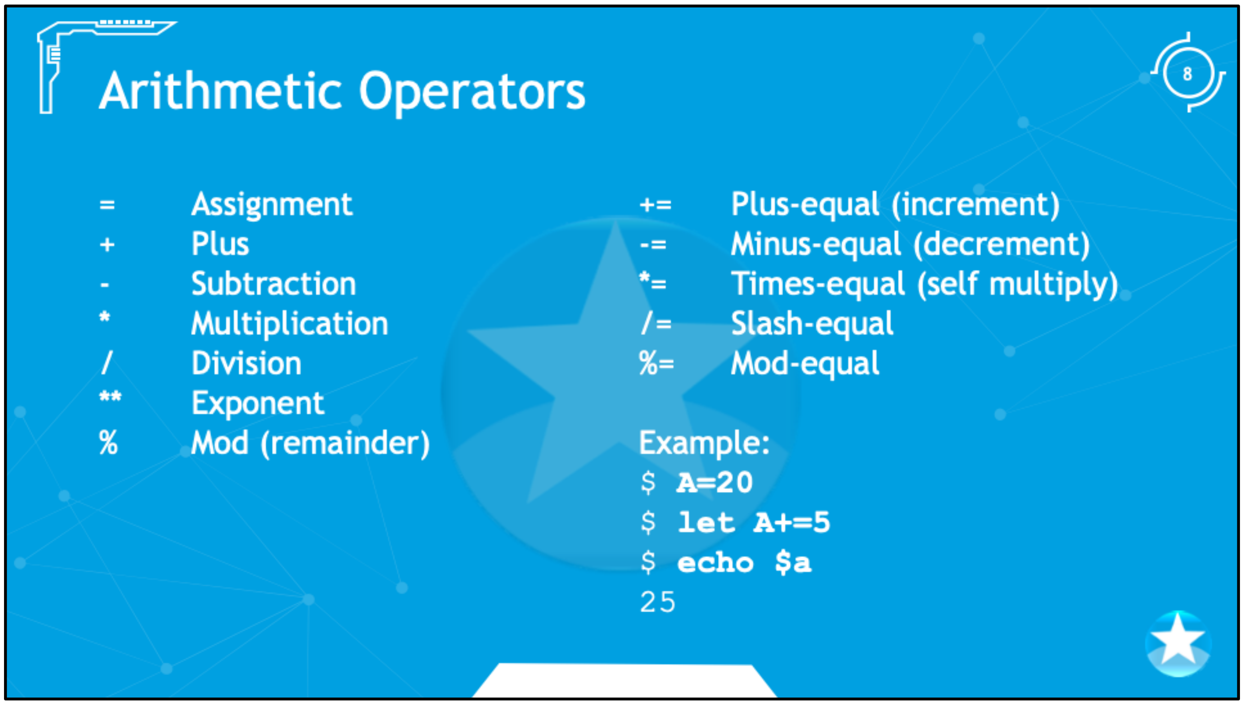
```
$ A=44
$ echo $A
44
$ B=$A+1
$ echo $B
44+1
$ let B=$A+1
$ echo $B
45
```

The "let" command allows you to do mathematic expressions in BASH. Without it, variables are treated as text. For example:

```
$ A=44
$ echo $A
44
$ B=$A+1
$ echo $B
44+1
```

If you want to evaluate a mathematic expression, you use the "let" command.

```
$ A=44
$ echo $A
44
$ let B=$A+1
$ echo $B
45
```



Arithmetic Operators

=	Assignment
+	Plus
-	Subtraction
*	Multiplication
/	Division
**	Exponent
%	Mod (remainder)

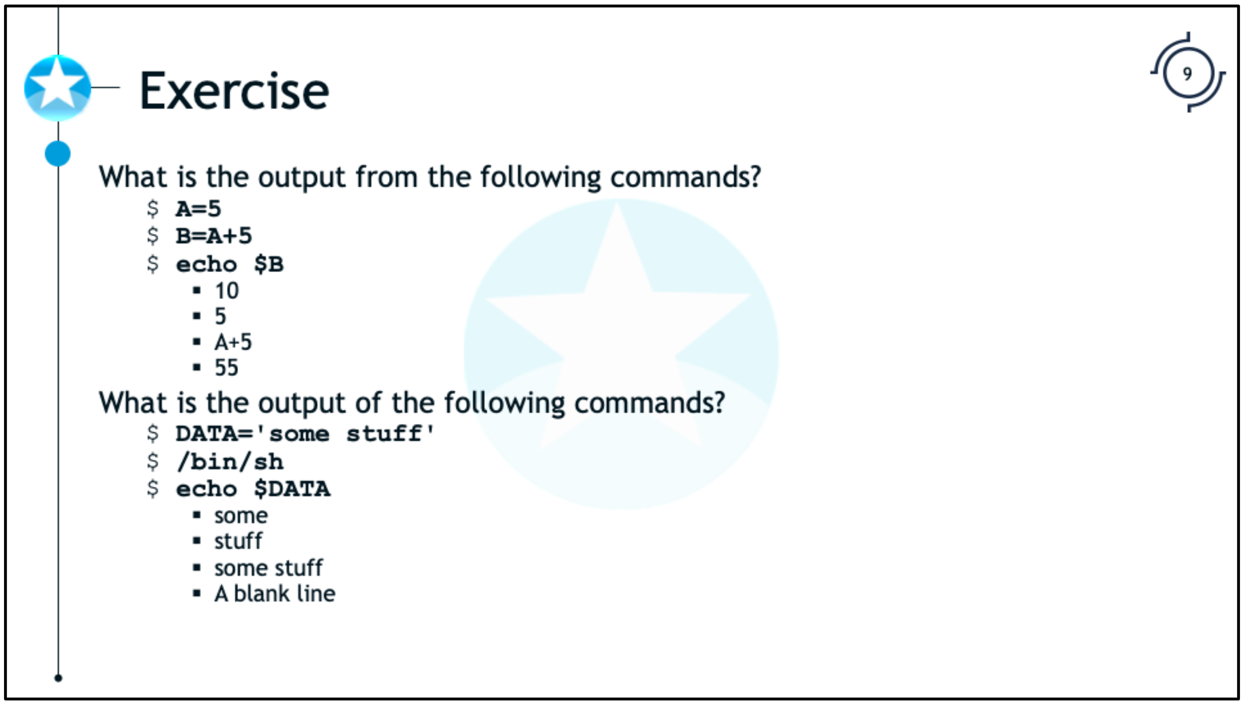
+=	Plus-equal (increment)
-=	Minus-equal (decrement)
*=	Times-equal (self multiply)
/=	Slash-equal
%=	Mod-equal

Example:

```
$ A=20
$ let A+=5
$ echo $a
25
```

The left column is pretty straightforward, but the right column is probably new to you. These operators will modify the variable itself. The command `A+=5` can be written as `A=A+5`. Here are examples of the different operators in the right column:

```
$ A=20
$ let A+=5
$ echo $A
25
$ A=20
$ let A-=5
$ echo $A
15
$ A=20
$ let A/=5
$ echo $A
4
$ A=20
$ let A%=6
$ echo $A
2
```



Exercise

9

What is the output from the following commands?

```
$ A=5
$ B=A+5
$ echo $B
```

- 10
- 5
- A+5
- 55

What is the output of the following commands?

```
$ DATA='some stuff'
$ /bin/sh
$ echo $DATA
```

- some
- stuff
- some stuff
- A blank line

What is the output from the following commands?

```
$ A=5
$ B=A+5
$ echo $B
```

a. 10

b. 5

c. A+5

d. 55

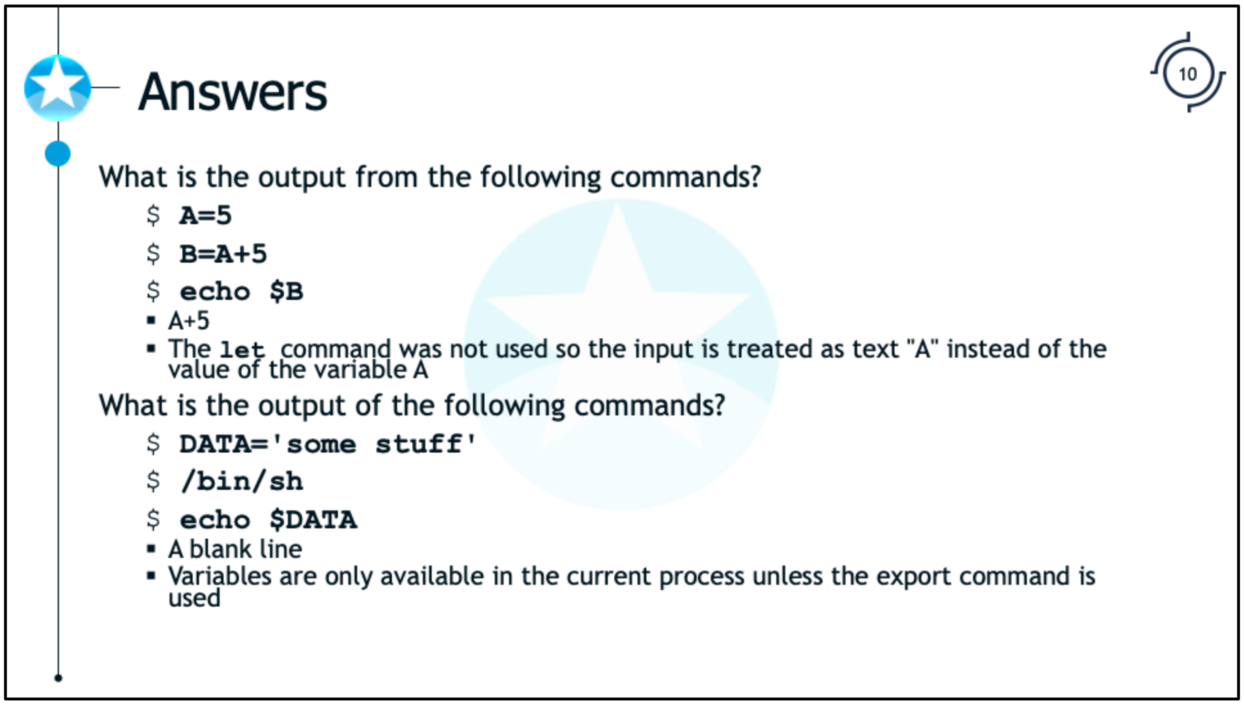
What is the output of the following commands?

```
$ DATA='some stuff'
$ /bin/sh
$ echo $DATA
```

a. some

b. stuff

c. some stuff
d. A blank line


Answers

What is the output from the following commands?

```
$ A=5
$ B=A+5
$ echo $B
```

- A+5
- The `let` command was not used so the input is treated as text "A" instead of the value of the variable A

What is the output of the following commands?

```
$ DATA='some stuff'
$ /bin/sh
$ echo $DATA
```

- A blank line
- Variables are only available in the current process unless the export command is used

What is the output from the following commands?

```
$ A=5
$ B=A+5
$ echo $B
```

Answer is C: `A+5`

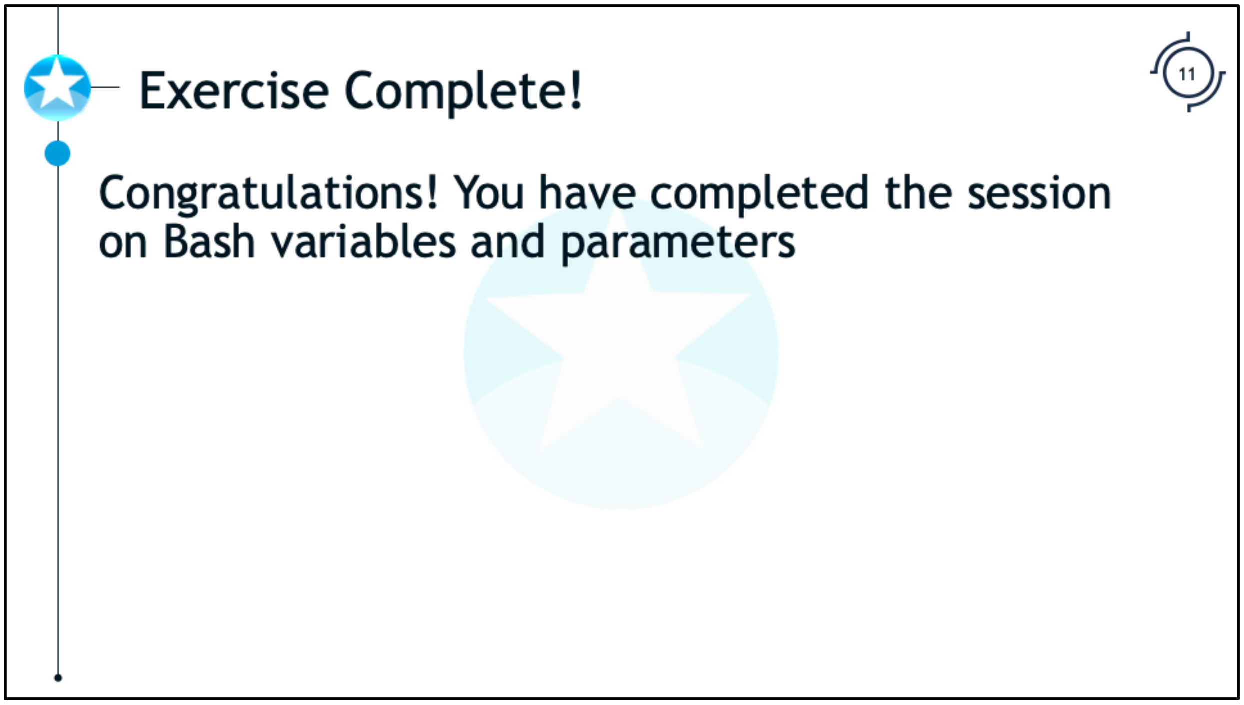
The let command was not used so the input is treated as text "A" instead of the value of the variable A

What is the output of the following commands?

```
$ DATA='some stuff'
$ /bin/sh
$ echo $DATA
```

Answer is D: A blank line

Variables are only available in the current process unless the export command is used

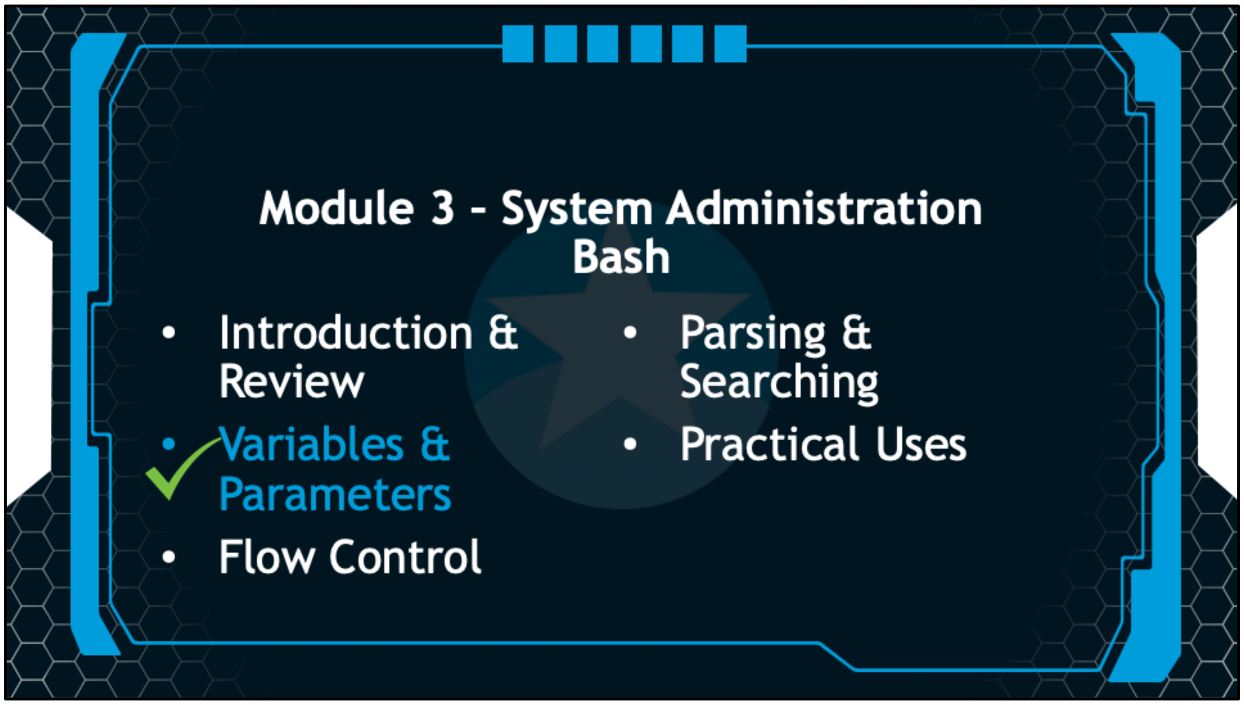


Exercise Complete!

Congratulations! You have completed the session on Bash variables and parameters

11

Congratulations! You have completed the session on Bash variables and parameters.



Module 3 - System Administration Bash

- Introduction & Review
- ✓ Variables & Parameters
- Flow Control
- Parsing & Searching
- Practical Uses

In the next session we will examine flow control in Bash.