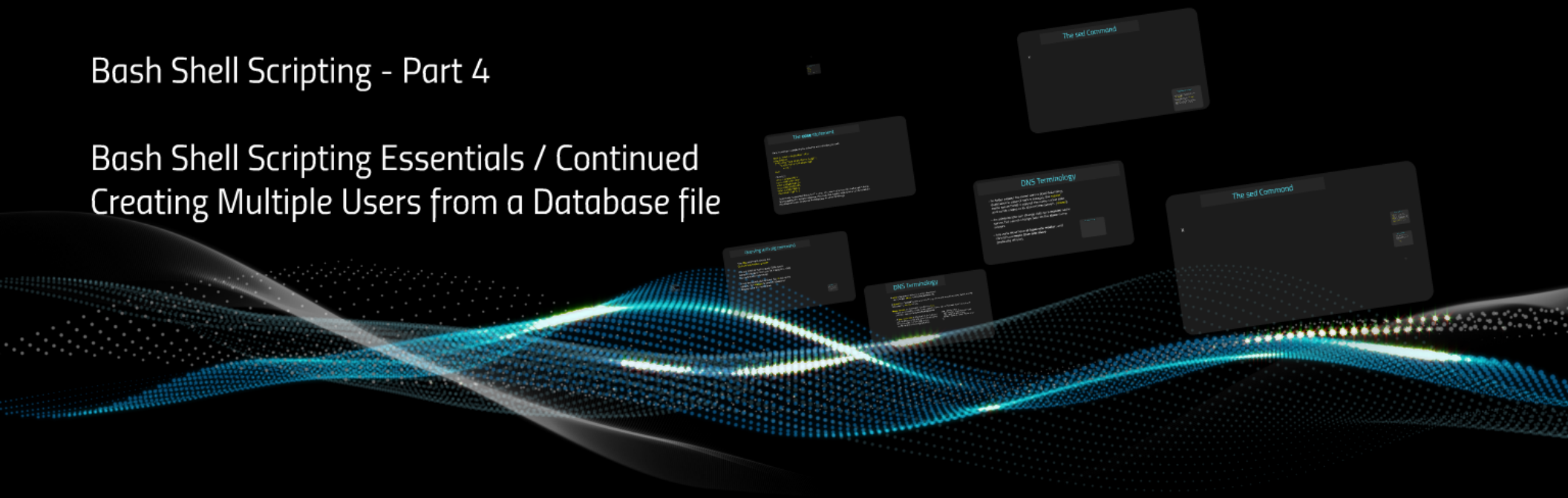
OPS235

Bash Shell Scripting - Part 4

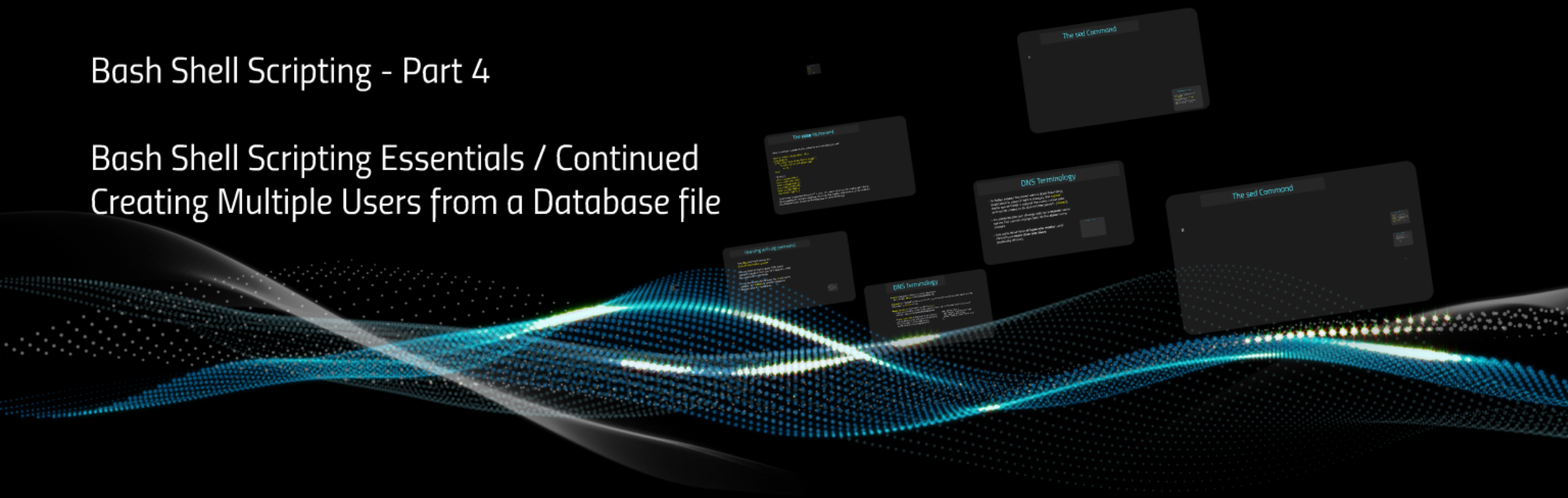
Bash Shell Scripting Essentials / Continued
Creating Multiple Users from a Database file



OPS235

Bash Shell Scripting - Part 4

Bash Shell Scripting Essentials / Continued
Creating Multiple Users from a Database file



Bash Shell Scripting - Using options

In labs1, 2 and 3 you have learned some useful tools to create useful Bash Shell scripts which included: variables, command substitution, mathematical operators, logic, loops, the Here Document and the sed command .

In Lab4, you will be learning to make your shell scripts work more like commands (i.e. commands that accept options). In order to do this, we need to learn 2 additional scripting tools:

- The case statement
  A logic statement that is considered to be a useful replacement for the if-elif-else command in certain situations.

- The getopts function
  A build in function used to verify if an option for that script is valid, and if the options includes data to use.

The **case** statement

The **case** statement is a control-flow statement that works in a similar way as the **if-elif-else** statement (but is more concise). This statement presents scenerios or "cases" based on values or regular expressions (not ranges of values like if-elif-else statements).

The statement **)** is used to represent a constant that will perform an action if the variable matches that value.

A break statement **;;** is used to "break-out" of the statement (and not perform other actions).

A default case ***** is also used to catch exceptions

The **case** statement

Here is an example:

```
read -p "pick a door (1 or 2): " pick
case $pick in
  1) echo "You win a car!" ;;
  2) echo "You win a bag of dirt!" ;;
  *) echo "Not a valid entry"
     exit 1 ;;
esac
```

OUTPUT:

```
pick a door (1 or 2): 7
Not a valid entry
pick a door (1 or 2): 1
You win a car!
```

Comments: Notice that this is NOT a loop. The user needs to re-run the command if the choice is not 1 or 2. This can be useful for error checking.

The **case** statement

Here is another example that is suited for error checking as well:

```
read -p "enter a single digit: " digit
case $digit in
  [0-9]) echo "Your single digit is: $digit" ;;
        *) echo "not a valid single digit"
           exit 1 ;;
esac
```

OUTPUT:

```
enter a single digit: x
not a valid single digit
enter a single digit: 23
not a valid single digit
enter a single digit: 5
Your single digit is: 5
```

Comments: Notice that this is NOT a loop. The user had to run this shell script 3 times before getting the correct response. Also note that regular expressions can be used for the different cases to provide flexibility (eg. for error checking)

The **getopts** function

The **getopts** function allows the shell scripter to create scripts that accept options (like options for Linux commands). This provides the Linux administrator with scripts that provide more flexibility and versatility. A built-in function called getopts (i.e. get command options) is used in conjunction with a while loop and a case statement to carry out actions based on if certain options are present when the shell script is run.

The variable $OPTARG can be used if an option accepts text (denoted in the getopts function with an option letter followed by a colon. Case statement exceptions use the **:)** and **\?)** cases for error handling.

The **getopts** function

Here is an example of using a while loop, a case statement and the getopts function to make a shell script only accept the options **-a**, **-b**, or **-c** (or any combination of those letters after the minus sign):

```
while getopts abc: name
do
  case $name in
    a) echo "Action for option \"a\"" ;;
    b) echo "Action for option \"b\"" ;;
    c) echo "Action for option \"c\""
       echo Value is: $OPTARG" ;;
    :) echo "Error: You need text after -c option"
       exit 1 ;;
    \?) echo "Error: Incorrect option"
       exit 1 ;;
esac
done
```

The **getopts** function

Take some time to view the code on the previous slide to understand how it works.

Some OUTPUT Examples (if this was a script called option.bash):
./option.bash -x
Error: Incorrect option

./option.bash -c
Error: You need text after -c

./option.bash -ab -c hello
Action for option "a"
Action for option "b"
Action for option "c
Value is: hello

The **getopts** function

In lab4, you will be creating a Bash shell script that will require an option **-i** followed by an existing pathname for a database text file that will automatically create users (from any number from 1 to 10,000)!

Try to understand how that shell script works and also try to gain an appreciation how creating a testing a shell script can help save a Linux system administrator time!

OPS235

Bash Shell Scripting - Part 4

Bash Shell Scripting Essentials / Continued
Creating Multiple Users from a Database file