**Shell Scripting for DevOps**

**Introduction & Shell Types**

* **What is Shell Scripting?**
  + Shell scripting is writing a series of commands for the shell to execute. The shell acts as a command-line interpreter between the user and the system. Shell scripts automate tasks, improve productivity, and are essential for DevOps.
  + **Example**: You can write a script that automatically backs up important files every day at midnight.
* **Types of Shells**:
  + **Bash (Bourne Again Shell)**: The most popular shell used in Linux. It is often the default shell in most distributions.
  + **Zsh (Z Shell)**: A more feature-rich shell that offers advanced autocompletion and customization.
  + **Fish (Friendly Interactive Shell)**: A user-friendly shell designed for ease of use.
  + **Ksh (Korn Shell)**: An older shell, often used for its scripting capabilities in enterprise environments.

**Shell Script Basics & First Script**

* **Creating Your First Script**:
  + To create a shell script, use a text editor to write your commands and save the file with the .sh extension.
  + **Example**: Create a script named hello.sh:

#!/bin/bash

echo "Hello, World!"

* **Making the Script Executable**:
  + Use the chmod command to make the script executable.

chmod +x hello.sh

* **Running the Script**:
  + Run the script by typing ./ followed by the script name:

./hello.sh

**Variables, Comments, File Naming**

* **Variables**:
  + Variables in shell scripts store data that can be used later. They do not need to be declared; just assign a value.
  + **Example**:

#!/bin/bash

name="DevOps Student"

echo "Welcome, $name!"

* **Comments**:
  + Comments are used to explain the code. They are preceded by a # symbol.
  + **Example**:

# This is a comment explaining the next line of code

echo "This will be printed."

* **File Naming**:
  + Script files should end with .sh to indicate it's a shell script, but this is not mandatory.
  + Example: backup.sh, deploy.sh, etc.

**Command Line Arguments & Escape Characters**

* **Command Line Arguments**:
  + You can pass arguments to scripts via the command line. These arguments can be accessed inside the script using $1, $2, etc., where $1 is the first argument, $2 is the second, and so on.
  + **Example**:

#!/bin/bash

echo "First argument: $1"

echo "Second argument: $2"

Running the script:

./script.sh arg1 arg2

* **Escape Characters**:
  + Escape characters are used to give special meaning to characters. The backslash (\) is the most common escape character.
  + **Example**:

echo "This is a line break.\nThis is the next line."

**String & Arithmetic Operations**

* **String Operations**:
  + String concatenation:

str1="Hello"

str2="World"

result="$str1 $str2"

echo $result # Outputs: Hello World

* **Arithmetic Operations**:
  + You can perform basic arithmetic in shell scripts.
  + **Example**:

#!/bin/bash

a=10

b=5

sum=$((a + b))

echo "Sum: $sum"

* + **Other Operations**:
    - Addition: sum=$((a + b))
    - Subtraction: diff=$((a - b))
    - Multiplication: prod=$((a \* b))
    - Division: quot=$((a / b))

**User Interaction using read**

* **User Input**:
  + The read command is used to get user input during the execution of the script.
  + **Example**:

#!/bin/bash

echo "Enter your name:"

read name

echo "Hello, $name!"

**Redirection, Pipes, Control Statements**

**Redirection:**

* **Output Redirection**: You can redirect the output of a command to a file using >.
  + **Example**:

echo "Hello World" > output.txt # Saves output to output.txt

* **Appending Output**: Use >> to append to a file.
  + **Example**:

echo "Appended text" >> output.txt # Appends text to output.txt

* **Input Redirection**: Redirect the content of a file to a command using <.
  + **Example**:

sort < input.txt # Sorts the content of input.txt

**Pipes:**

* **Pipes (|)**: The pipe sends the output of one command to the input of another command.
  + **Example**:

cat file.txt | grep "keyword" # Sends file.txt content to grep to search for "keyword"

**Control Statements**

* **if Statement**:
  + Used for conditional execution of commands.
  + **Example**:

#!/bin/bash

echo "Enter a number:"

read num

if [ $num -gt 10 ]; then

echo "Number is greater than 10."

else

echo "Number is 10 or less."

fi

* **for Loop**:
  + Used for iterating over a sequence.
  + **Example**:

#!/bin/bash

for i in {1..5}

do

echo "Iteration $i"

done

* **while Loop**:
  + Executes commands as long as a condition is true.
  + **Example**:

#!/bin/bash

count=1

while [ $count -le 5 ]

do

echo "Count: $count"

((count++))

done

* **switch-case**:
  + Used for multi-way branching.
  + **Example**:

#!/bin/bash

echo "Enter a number (1-3):"

read num

case $num in

1)

echo "You selected One."

;;

2)

echo "You selected Two."

;;

3)

echo "You selected Three."

;;

\*)

echo "Invalid selection."

;;

esac

**Functions**

* **Functions in Shell Scripting**:
  + Functions allow you to group commands together and reuse them.
  + **Example**:

#!/bin/bash

function greet() {

echo "Hello, $1!"

}

greet "Alice" # Outputs: Hello, Alice!

**Conclusion**

This comprehensive breakdown of Shell Scripting provides your students with both foundational knowledge and practical examples. By practicing these commands and writing scripts, they'll be able to automate tasks, interact with users, and build powerful automation tools used in DevOps pipelines.