LESSON NOTES

Intro to Linux

Scripting, Containers, and Automation

3.1.1 Shell Script Elements Part 1

Lesson Overview:

Students will:

- Understand some of the basic shell scripting commands

Guiding Question: How can shell scripts be used to automate common tasks?

Suggested Grade Levels: 9 - 12

Technology Needed: None

CompTIA Linux+ XK0-005 Objective:

3.1 - Given a scenario, create simple shell scripts to automate common tasks

- Shell script elements
  - Loops
    - while
    - for
    - until
  - Conditionals
    - if
    - switch/case
  - Shell parameter expansion
    - Globbing
    - Brace expansions
  - Comparisons
    - Arithmetic
    - String
    - Boolean
  - Variables
  - Search and replace
  - Regular expressions

This content is based upon work supported by the US Department of Homeland Security's Cybersecurity & Infrastructure Security Agency under the Cybersecurity Education Training and Assistance Program (CETAP).

Shell Script Elements

In shell scripting, there are many loops, conditionals, variables, and functions. We will investigate some of these with a brief definition followed by a screenshot generalization to explain how they function.

A while loop executes a set of commands as long as a specified condition is true.

```
while [ condition ]; do
     commands
done
```

A for loop iterates over a sequence, like a range of numbers, and performs commands.

```
for variables in {start...end...step}; do
     commands
done
```

An until loop executes a set of commands as long as the specified condition is false.

```
until [ condition ]; do
     commands
done
```

Conditionals in shell scripts are control structures that allow you to make decisions and execute different sets of commands based on whether a certain condition is true or false.

An if statement executes a set of commands based on the evaluation of a condition.

```
if [ condition ]; then
     commands
fi
```

Switch/Case provides multiple possible execution paths based on the value of an expression.

```
case $variable in
     pattern1)
          commands
          ;;
     pattern2
          commands
          ;;
     *)
          default commands
          ;;
esac
```

Shell parameter expansion is a feature in shell scripting that allows you to manipulate and expand the values of variables. It provides a convenient way to perform various operations on variables, such as extracting substrings, performing pattern matching, and more. Parameter expansion is commonly used in shell scripts to enhance the flexibility and efficiency of variable handling.

Globbing matches filenames with patterns, similar to regular expressions.

```
files=*.txt
```

Brace expansions generate arbitrary strings using curly braces.

```
echo {1..5}
```

In shell scripting, comparisons are essential for making decisions based on the values of variables or the success/failure of commands.

Perform arithmetic operations within double parentheses.

```
if (( num1 > num2 )); then
     commands
fi
```

Compare strings using operators like == (the same) or != (not the same).

```
if [ "$str1" == "$str2" ]; then
     commands
fi
```

In shell scripting, there isn't a native boolean data type like in some other programming languages. However, boolean logic is still implemented using the exit status of commands and conditional

statements. Users can combine conditions using logical operators (&& for AND, || for OR).

```
if [ condition1 ] && [ condition2 ]; then
     commands
fi
```

In shell scripting, variables are used to store and manipulate data. They act as placeholders for values that can be referenced or modified within a script.

```
variable_name="value"
```

If there is a piece of text that needs to have a certain part replaced, "search and replace" does just that.

```
result=${string/old/new}
```

Regular expressions (regex or regexp) are powerful patterns used for matching character combinations within strings. They are widely used in shell scripting for tasks like string manipulation, text parsing, and pattern matching.

These are the basic building blocks for shell scripting. They allow you to create powerful and dynamic scripts for automating tasks on the command line.