LESSON NOTES

Intro to Linux

Scripting, Containers, and Automation

3.1.2 Shell Script Elements Part 2

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

- Standard stream redirection
  - |
  - ||
  - >
  - >>
  - <
  - <<
  - &
  - &&
  - Redirecting
    - stderr
    - stdout
- Here documents
- Exit codes
- Shell built-in commands
  - read
  - echo
  - source

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

The **|** (Pipe) connects the output of one command as the input to another command. This is a way to chain commands together, allowing the output of the first command to be processed by the second. For example, **`fortune | cowsay`** would generate a fortune and use the tool cowsay to display the fortune (if both fortune and cowsay were installed).

The **||** (OR) command executes the command following it ONLY IF the preceding command fails/returns an error. This is often used in control flow. This can be used as an alternative to an if-then-else command.

The **>** (Output Redirection) redirects the standard output to a file. If the file already exists, it will be overwritten. If the file doesn't exist, it will be created. For example, **`cat example.txt > output.txt`** would take the text contents of the example.txt file and write that to the output.txt file.

The **>>** (Output Redirection, too) command is similar to **>**, but it appends the standard output to the end of the file instead of overwriting it. This can be used for note taking or creating your own log file.

The **<** (Input Redirection) command redirects the input from a file to a command. The command will take its input from the specified file. For example, **`grep "pattern" < input.txt`** would search for a specific pattern in the input.txt file.

The **<<** (Here Document) allows multiple lines to be put into a command. This is often used in scripts to provide input to a command in a script. In general, the command would look like the following:

```
command << EOF
Line 1
Line 2
EOF
```

The **&>** (Ampersand) combines standard output and standard error to a specified file. If the file already exists, it is overwritten, otherwise, create the file. Don't confuse this with **&** which makes a command run in the background.

The **&&** (AND) command executes the command following it ONLY IF the preceding command succeeds. This is also used in control flow.

The preceding commands are part of control flow and redirection. STDOUT is the standard output stream where a program writes its regular output. By default, it's the terminal or console where the program is run. For example, if you run a command like ls in a terminal, the list of files and directories is sent to STDOUT. STDERR is the standard error stream where a program writes its error messages. It's also sent to the terminal or console by default, but it's meant to separate error messages from regular output. If there's an issue with a command, the error message will be displayed on STDERR.

These streams are helpful for debugging and understanding what's happening when you run a command or a program. They allow you to capture and redirect output and errors as needed. For example, you might want to redirect the output of a command to a file using **>** or **>>**, or you might want to separate regular output from error messages by redirecting each to a different file or location.

Exit Codes

Exit codes, also known as return codes or status codes, are numeric values returned by a command or a program to indicate the result of its execution. In most systems, including Unix-like operating systems (Linux, macOS) and Windows, an exit code of 0 typically signifies success, while non-zero values indicate an error or some kind of failure.

When a command or a script is run, the OS or shell keeps track of the exit code of that command. This allows other scripts or processes to check whether the preceding command was successful or encountered an issue. Here are some common conventions for exit codes:

- 0: Success or no error.
- 1-255: Indicates an error or some specific condition depending on the program.

It's common for programs to use different exit codes to signify different types of errors or issues. For example, a script might return different exit codes to indicate whether a required file is missing, a network connection failed, or a user input was invalid.

Shell Built-In Commands

Shell built-in commands are commands that are built into the shell itself, rather than being separate executable files. This makes them more efficient because they are part of the shell's functionality and don't require a separate process to be spawned. The following are a few of the built-in commands.

`read` is used to read input from the user or from a file. It reads a line from the standard input (usually the keyboard) and assigns the words to variables. echo is used to print text or variables to the standard output (usually the terminal). source (or its equivalent) is used to execute commands from a file in the current shell environment. This is commonly used to run shell scripts or to set environment variables.