

Lab - Linux Fedora BASH Scripting Part I

Introduction

One of the great features of Linux is writing BASH scripts. Compared to writing Windows batch files, BASH scripting is much more flexible and comes with advanced features you can't find in a batch script. To understand how the BASH shell works, you must understand the logic of how Linux is built.

Simply put, BASH (Bourne-Again Shell) is the default shell we are provided with every Linux distribution. It is the command line interpreter (CLI) for GNU (GNU's Not Unix) operating system. When we open a terminal session in Linux, we are using the BASH shell. Though GNU operating system provides different shells, BASH is the default out of the box shell for Linux.

A shell is a program that provides the command line (i.e., the all-text display user interface) on Linux and other Unix-like operating systems. It also executes (i.e., runs) commands typed into the terminal and displays the results. BASH is the default shell on Linux.

The GNU project is a mass collaborative initiative for the development of free software. Richard Stallman founded the project in 1978 at MIT.

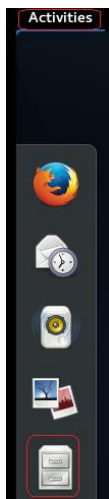
When we type any syntax into a terminal window, it is the job of the BASH shell to interpret and execute the command. The BASH shell is in the **bin** folder which is located at the root of the operating system. To access the root of any Linux system, we use the Unix command of: /

Let's create a visual of accessing the bin folder located at the root in Fedora.

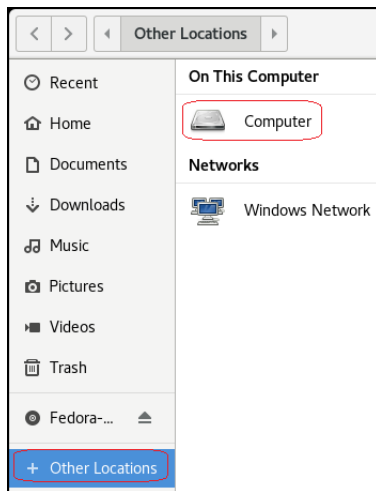
Start the Lab

Launch your Fedora Workstation virtual machine.

From the Fedora desktop, click **Activities** and from the Application Quick Launch, open the link for **Files**. You are now at the root for Kali or Ubuntu or any other GNU operating system.



At the bottom of the file browser, click on **Other Locations** and click on **Computer**.



Fedora has a shortcut for the bin folder located at its root. In Fedora, the actual location is in the `USR` folder, but in most flavors of Linux, you will find it at the root.

In this lab, students will be introduced to BASH scripting.

What is a Script?

Scripts are collections of commands, stored in a file. The shell can read this file and act on the commands as if they were typed at the keyboard. The shell also provides a variety of useful programming features to make scripts truly powerful.

What is a Shell?

A shell is a user program or some environment provided for user interaction. Shell is a command language interpreter that executes commands read from the standard input device (keyboard) or from a file. Shell is not part of system kernel but uses the system kernel to execute programs, create files, etc.

Types of Shells

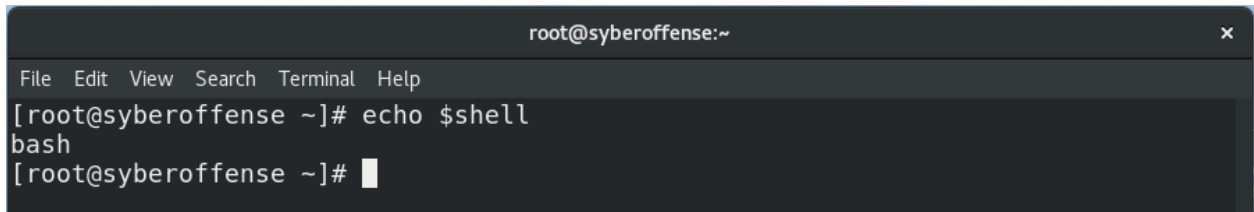
Bourne shell: The Bourne shell, written by Stephen Bourne at Bell Labs, was one of the major shells used in early versions of Linux and is the de facto standard. Every Unix-like system has at least one shell compatible with the Bourne shell. The Bourne shell program name is “sh” and it is typically located in the file system hierarchy at `/bin/sh`.

A Shell script is a simple text file with “.sh” extension, having executable permission.

BASH stands for **Bourne Again Shell**.

To see what default shell Fedora is using, we open a terminal and at the prompt type:

echo \$SHELL

A terminal window titled 'root@syberoffense:~' with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is '[root@syberoffense ~]#'. The command 'echo \$SHELL' is entered, and the output 'bash' is displayed. The prompt returns to '[root@syberoffense ~]#' with a cursor.

Note: I am logged on as root.

In Linux, the dollar sign (\$) stands for a shell variable. In this case, it means 'whatever.'

The 'echo' command returns 'whatever' you type in.

Linux commands have their own syntax and Linux won't forgive your mistakes. If you fat finger a command, you won't damage anything, but do not expect Linux to figure out what it is you want unless you input the right command syntax.

At the beginning of every script, we must give instruction on what shell to use and where the shell program is located. The first line is important!

#!/bin/bash – This is called a shebang. It is written at the top of a shell script, and it passes the instruction to the program /bin/bash.

Since the BASH interpreter is in the same folder as nearly all the default utilities and tools built into the GNU operating system, it makes it very easy for the completion of commands and the running of scripts using the CLI.

For instance, to create and edit scripts, we need a script editor.

There are default text editors that come with Fedora and VI is one. For this lab, we'll be using NANO as the text editor for creating our BASH scripts.

Install Nano

Install the package using yum. At the terminal, type:

yum -y install nano

```
root@syberoffense:~  
File Edit View Search Terminal Help  
[root@syberoffense ~]# yum -y install nano  
Fedora 26 - x86_64 - Updates      851 kB/s | 15 MB    00:18  
RPM Fusion for Fedora 26 - Free - Updates 12 kB/s | 250 kB    00:20  
Last metadata expiration check: 0:00:00 ago on Tue 03 Oct 2017 04:22:22 PM +08.  
Package nano-2.8.4-1.fc26.x86_64 is already installed, skipping.  
Dependencies resolved.  
Nothing to do.  
Complete!  
[root@syberoffense ~]#
```

We can use the **whereis** command to locate nano.

```
root@syberoffense:~  
File Edit View Search Terminal Help  
[root@syberoffense ~]# whereis nano  
nano: /usr/bin/nano /usr/share/nano /usr/share/man/man1/nano.1.gz /usr/share/info/nano.info.gz  
[root@syberoffense ~]#
```

Writing our first script...

We'll use **nano** as our default text editor for this lab. We begin by open **nano** from the terminal and telling to create text file named: **hello.sh**

Type: **nano hello.sh**

```
root@syberoffense:~  
File Edit View Search Terminal Help  
[root@syberoffense ~]# nano hello.sh
```

Nano opens our 'hello.sh' text file!

On the first line type:

#!/bin/bash

This is the convention used so the Linux shell knows what interpreter to run. If this script was written in Python or Pearl, the word bash, would be replaced with the name programming language so the script could be interpreted correctly.

The # sign and the ! at the beginning of the first line is called the shabang,

Hit enter

On the next line, type a comment describing what this script is used for:

```
# My first script
```

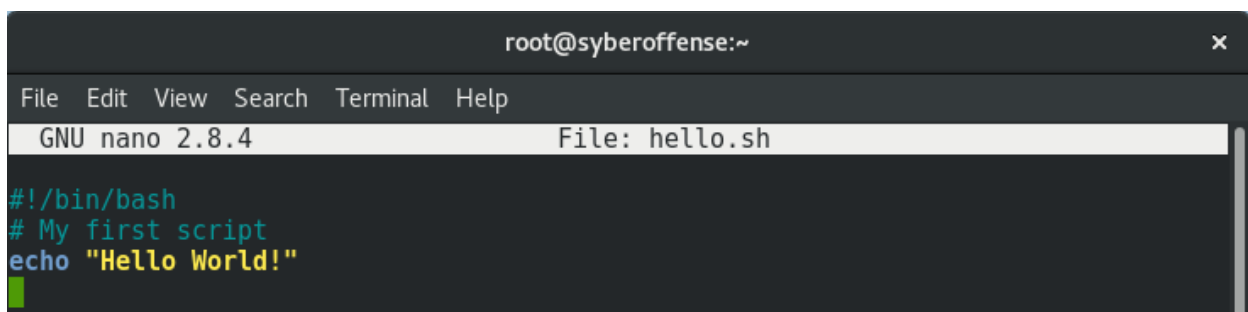
Hit enter

On the next line type:

```
echo "Hello World!"
```

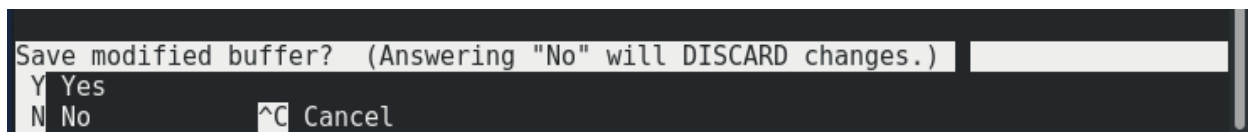
Hit enter.

Here's what we have so far:



```
root@syberoffense:~  
File Edit View Search Terminal Help  
GNU nano 2.8.4 File: hello.sh  
#!/bin/bash  
# My first script  
echo "Hello World!"
```

Enter 'ctrl+x' to close the editor. When asked to save the modified buffer, enter 'Y' for yes.



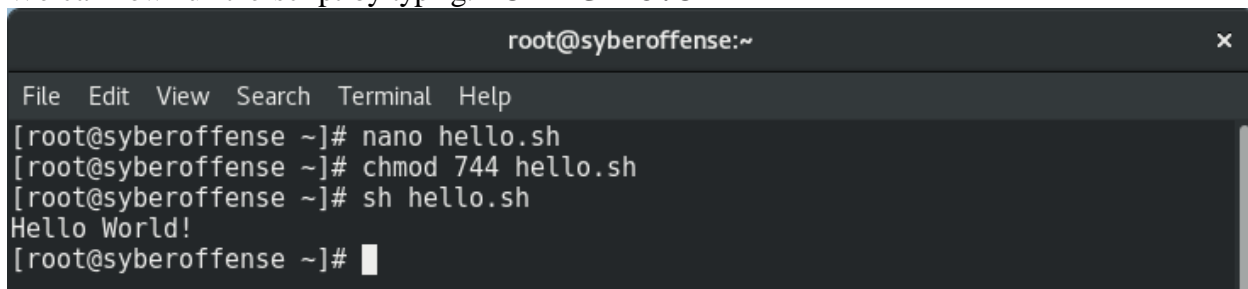
```
Save modified buffer? (Answering "No" will DISCARD changes.)  
Y Yes  
N No ^C Cancel
```

Select the default for the file name to write using the enter key. This brings you back to the terminal prompt.

We now need to make the hello.sh file an executable. At the prompt type:

```
chmod 744 hello.sh
```

We can now run the script by typing: **sh hello.sh**



```
root@syberoffense:~  
File Edit View Search Terminal Help  
[root@syberoffense ~]# nano hello.sh  
[root@syberoffense ~]# chmod 744 hello.sh  
[root@syberoffense ~]# sh hello.sh  
Hello World!  
[root@syberoffense ~]#
```

Let's review what we wrote.

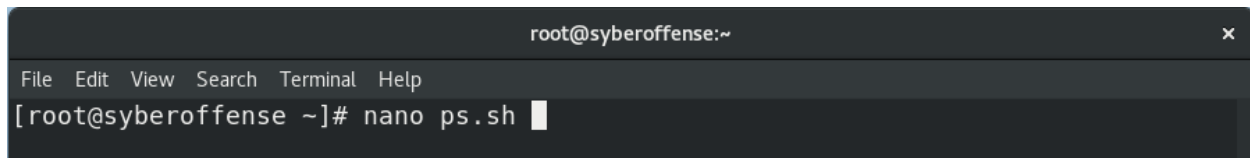
```
#!/bin/bash (the shebang)
# My first script (Our comment, anything following a '#' sign is
a comment)
echo "Hello World!" (is the main part of this script)
```

A very simple script but we're about to make it more interesting. Let write a script that checks for processes running.

Using what we know so far, write the following BASH script:

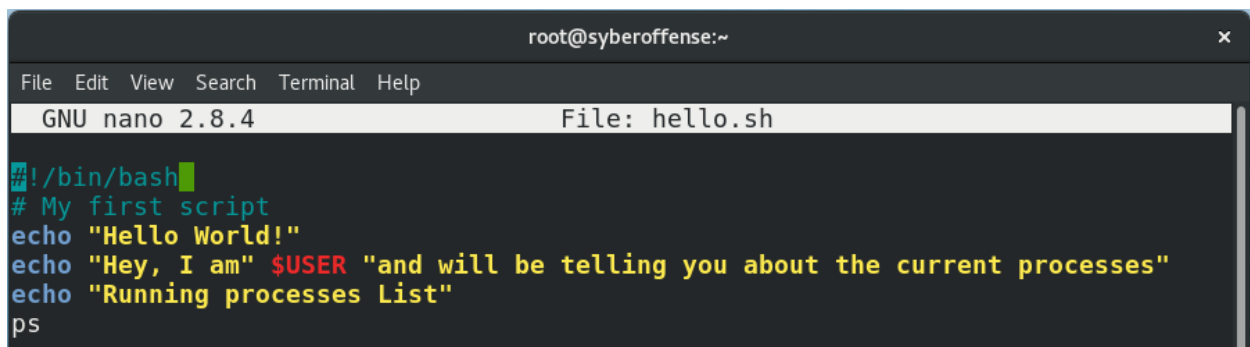
```
#!/bin/bash
echo "Hello $USER"
echo "Hey, I am" $USER "and will be telling you about the
current processes"
echo "Running processes List"
ps
```

1. Using nano, open a new text file and name it 'ps.sh.'



```
root@syberoffense:~
File Edit View Search Terminal Help
[root@syberoffense ~]# nano ps.sh
```

2. Type in the commands one line at a time.



```
root@syberoffense:~
File Edit View Search Terminal Help
GNU nano 2.8.4 File: hello.sh
#!/bin/bash
# My first script
echo "Hello World!"
echo "Hey, I am" $USER "and will be telling you about the current processes"
echo "Running processes List"
ps
```

3. Save the file using ctrl+x.
4. Save the modifier buffer by entering 'Y'
5. Press enter to save the file using the name 'ps.sh'
6. Make the file executable: **chmod 744 ps.sh**
7. Run the script: **sh ps.sh**

```
root@syberoffense:~  
File Edit View Search Terminal Help  
[root@syberoffense ~]# nano hello.sh  
[root@syberoffense ~]# sh hello.sh  
Hello World!  
Hey, I am root and will be telling you about the current processes  
Running processes List  
  PID TTY          TIME CMD  
 2464 pts/0    00:00:00 bash  
 3618 pts/0    00:00:00 sh  
 3619 pts/0    00:00:00 ps  
[root@syberoffense ~]# nano hello.sh  
[root@syberoffense ~]#
```

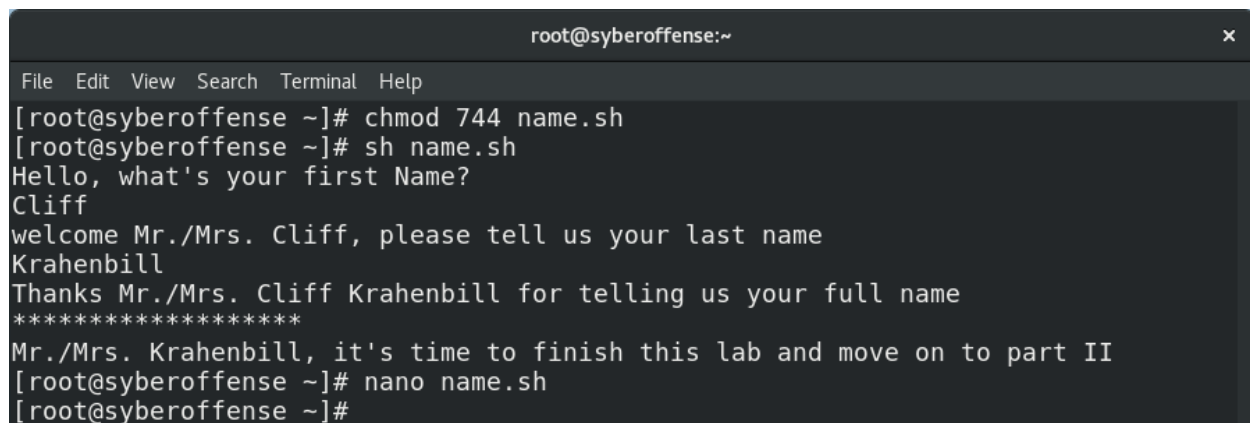
In this third script, we add in some interaction. With what we have learned so far create a BASH script using the following commands:

```
#!/bin/bash  
echo "Hello, what's your first Name?";  
read a;  
echo "welcome Mr./Mrs. $a, please tell us your last name";  
read b;  
echo "Thanks Mr./Mrs. $a $b for telling us your full name";  
echo "*****"
```

1. Create a nano file. Name it something you can remember (name.sh)
2. Type in the above commands, one line at a time.

```
root@syberoffense:~  
File Edit View Search Terminal Help  
GNU nano 2.8.4 File: name.sh  
#!/bin/bash  
echo "Hello, what's your first Name?";  
read a;  
echo "welcome Mr./Mrs. $a, please tell us your last name";  
read b;  
echo "Thanks Mr./Mrs. $a $b for telling us your full name";  
echo "*****"  
echo "Mr./Mrs. $b, it's time to finish this lab and move on to part II"
```

3. Save the file using ctrl+x.
4. Save the modifier buffer by entering 'Y'
5. Press enter to save the file using the name 'name.sh'
6. Make the file executable: **chmod 744 name.sh**
7. Run the script: **sh name.sh**

A terminal window titled 'root@syberoffense:~' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows a BASH script being executed. The script prompts for a first name, then a last name, and then a full name. It uses the 'read' command for input and 'echo' for output. The script ends with a message about moving on to part II and then runs 'nano name.sh' and returns to the prompt.

```
root@syberoffense:~  
File Edit View Search Terminal Help  
[root@syberoffense ~]# chmod 744 name.sh  
[root@syberoffense ~]# sh name.sh  
Hello, what's your first Name?  
Cliff  
welcome Mr./Mrs. Cliff, please tell us your last name  
Krahenbill  
Thanks Mr./Mrs. Cliff Krahenbill for telling us your full name  
*****  
Mr./Mrs. Krahenbill, it's time to finish this lab and move on to part II  
[root@syberoffense ~]# nano name.sh  
[root@syberoffense ~]#
```

Summary

In lab 1 of this two-part lab series, we learned about BASH file scripting and how the process ties in so easily with the default BASH shell used by most Linux operating system. We saw how to use the echo command to repeat what we type in. We learned how to comment out a line for informational purposes. We learned how to use a text editor to create a script. We learned how to make a file an executable and how to run the script from a terminal.

We saw how we could use the read command to create some type of interaction between the user and the script. We learned how to run a script to query the system. With just slight modification, we can get these scripts to do even more.

In Lab 2, we take BASH scripting to the next level.

End of Lab 1!