

ULI101: INTRODUCTION TO UNIX / LINUX AND THE INTERNET

WEEK 10 LESSON 1

INTRODUCTION TO SHELL SCRIPTING / CREATING SHELL SCRIPTS / SHELL VARIABLES

PHOTOS AND ICONS USED IN THIS SLIDE SHOW ARE LICENSED UNDER [CC BY-SA](#)



LESSON I TOPICS

Shell Scripts

- Definition / Purpose
- Considerations When Creating Shell Scripts /
- Comments / She-bang line / **echo** command
- Creating Shell Scripts / Running Shell Scripts / Demonstration

Shell Variables

- Definition / Purpose
- Environment Variables / User Defined Variables / **read** command
- Demonstration

Perform Week 10 Tutorial

- Investigation I
- Review Questions (Questions Part A 1 – 2 , Part B **Walk-Thru #1**)

CREATING SHELL SCRIPTS

Definition

A **shell script** is a computer **program** designed to be run by the Unix **shell**, a **command-line interpreter**.

Typical operations performed by shell scripts include **file manipulation, program execution, and printing text**.

Reference: https://en.wikipedia.org/wiki/Shell_script

```
#!/bin/sh
set -ef

if test -n "$KSH_VERSION"; then
  puts() {
    print -r -- "$*"
  }
else
  puts() {
    printf '%s\n' "$*"
  }
fi

while getopts a whichopts
do
  case "$whichopts" in
    a) ALLMATCHES=1 ;;
    ?) puts "Usage: $0 [-a] args"; ;;
  esac
done
```

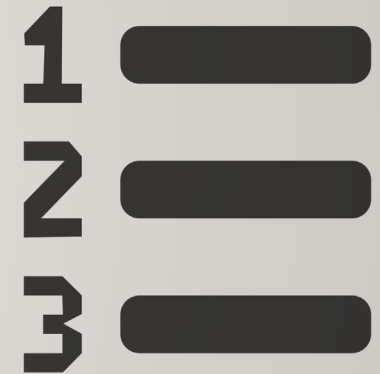
CREATING SHELL SCRIPTS

Considerations When Creating Shell Scripts

The reason to create shell scripts is to **automate** the execution of commonly issued **Linux commands, shell operations, math calculations** as well as **Logic / Loop** operations.

Prior to the creation of the shell script file, you should **plan** the shell script and **list steps** that you want to accomplish.

Those **sequence** of steps can then be used to create your shell script.



CREATING SHELL SCRIPTS

Considerations When Creating Shell Scripts

Once you have **planned** your shell script you need to **create** a **shell script file** via a **text editor** that will contain Linux commands.

When creating a shell script, avoid using filenames of **existing** Linux commands. You can use the **which** command to see if the filename is recognized as a Unix/Linux command: (e.g. `which shell-script-name`)

Adding an **extension** to your shell script filename will help to **identify** the type of shell that the shell script was designed to run.

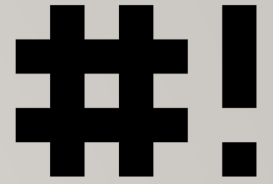
Examples:

`clean-directory.bash`

`copy-directory-structure.csh`



CREATING SHELL SCRIPTS



The Shebang Line

The # symbol makes the shell ignore text after this symbol so that text can be used to provide information of how the shell script works.

```
# This is a comment
```

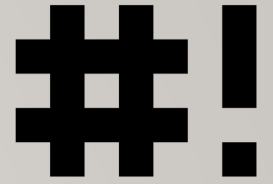
The **she-bang** line is a **special comment** at top of your shell script to run a shell script within a specific shell.

Example:

```
#!/bin/bash
```

The shebang line must appear on the **first** line and at the **beginning** of the line, otherwise, it will be treated as a **regular comment** and **ignored**.

CREATING SHELL SCRIPTS



The Shebang Line

Since Linux shells have evolved over a period of time, using a **she-bang line** forces the shell script to run in **a specific shell**, which could **prevent errors** in case an older shell does not recognize newer features from recent shells.

You can use the **which** command to determine the **full pathname** of the shell.

```
which bash  
/bin/bash
```


CREATING SHELL SCRIPTS

Displaying Text with the `echo` Command

When creating shell scripts, it is useful to **display text** to prompt the user for data, display results or notify the user of incorrect usage of the shell script.

The `echo` command is used to display text.

To prevent problems with special characters, it is recommended to use **double-quotes** which will allow the values of variables to be displayed.

Example:

```
echo "My username is: $USER"
```



```
echo hello
hello

echo 'hello'
hello

echo 'My username is: $USER'
My username is: $USER

echo "My username is: $USER"
My username is: murray.saul
```


RUNNING A SHELL SCRIPT

Running Shell Scripts

In order to run your shell script by name, you need to first assign **execute permissions** for the user.

To run your shell script, you can issue the shell script's pathname using a **relative**, **absolute**, or **relative-to-home** pathname

Examples:

```
chmod u+x myscript.bash
```

```
./myscript.bash
```

```
/home/username/myscript.bash
```

```
~/myscript.bash
```

FYI: You can **run** a shell script without **execute permissions** by issuing the **shell command** followed by the shell script's pathname.

Example:

```
bash ~murray.saul/scripts/week10-check-1
```

You can add the **current directory** that contains the shell script so it can be issued only by **filename** (not pathname).

Example:

```
PATH=$PATH:.
```

To be **persistent** on new shell instances, setting the PATH environment variable would need to be added in your **profile** (start-up) file (discussed in a later lesson).

INSTRUCTOR DEMONSTRATION

Task:

Create a Bash Shell script to clear the screen and then display all users that are currently logged onto the system.



SHELL SCRIPTING

Variables

Variables are used to **store information** to be referenced and manipulated in a computer program. They also provide a way of labeling data with a descriptive name, so our programs can be understood more clearly by the reader and ourselves...

...It is helpful to think of variables as **containers** that hold information. Their sole purpose is to label and store data in memory. This data can then be used throughout your program.

Reference: <https://launchschool.com/books/ruby/read/variables>



SHELL SCRIPTING

Using Variables

Shell variables are classified into **two groups**:

System (shell) variables:

Describes the OS system's **working environment** which can be used in a shell script.

User-created variables:

Customized variables **created by the programmer** for use in a shell script.

The name of a variable can be any sequence of **letters** and **numbers**, but it must **NOT begin with a number!**



SHELL SCRIPTING

Environment Variables

Shell **environment** variables define the **working environment** while in your shell. Some of these variables are displayed in the table below and its value can be viewed by issuing the following pipeline command: `set | more`

Variable Name	Purpose
PSI	Primary shell prompt
PWD	Absolute path of present working directory
HOME	Absolute path to user's home
PATH	List of directories where commands / programs are located
HOST	Host name of the computer
USER	Name of the user logged in
SHELL	Name (type) of current shell used

SHELL SCRIPTING

Environment Variables

Placing a dollar sign **\$** before a **variable name** will cause the variable to **expand** to the value contained in the variable.

Examples:

```
echo "My current location is: $PWD"  
who | grep $USER  
echo $HOST
```

```
echo "My current location is: $PWD"  
My current location is: /home/murray.saul  
  
who | grep $USER  
murray.saul pts/0          Jun 14 08:38 (99.236.168.165)  
  
echo $HOST  
matrix
```

SHELL SCRIPTING

User Defined (Created) Variables

***User-defined variables** are **variables** which can be **created** by the **user** and exist in the session.*

Reference: <https://mariadb.com/kb/en/user-defined-variables/>

You assign a value by using the **equal** sign (without spaces)

`name=value`

If a variable's value contain spaces or tabs,
it should be surrounded by **quotes**

`fullName="David G Ward"`

SHELL SCRIPTING

User Defined Variables

There are a few methods to remove a variable's value:

```
variableName=
```

or

```
unset variableName
```

Examples:

```
customerName=  
unset userAge
```

```
customerName=ACME  
echo $customerName  
ACME
```

```
customerName=  
echo $customerName
```

```
userAge=57  
echo $userAge  
57  
unset userAge  
echo $userAge
```

CREATING SHELL SCRIPTS

Prompting User for Input to Store in a Variable:

The `echo` command with the `-n` option will display text without the **newline** character.

The `read` command pauses and waits for a user to enter data and then stores the enter data into a **variable** when the user presses the **ENTER** key.

Example:

```
echo -n "Enter your age: "  
read age  
echo "Your age is $age"
```

For **Bash shell scripts**, the `read` command with the `-p` option prompts the user for data without requiring the `echo` command.

Example:

```
read -p "Enter your age: " age  
echo "Your age is $age"
```



```
echo -n "Enter your age: " ; read age  
Enter your age: 57  
  
echo "Your age is $age"  
Your age is 57  
  
read -p "Enter your age: " age  
Enter your age: 57  
echo "Your age is $age"  
Your age is 57
```

SHELL SCRIPTING

User Defined (Created) Variables

Issuing the **readonly** command after setting the variable's value **prevents** the user from changing the value of the variable while the shell script is running or during the duration of your shell session.

Examples:

```
readonly name  
readonly phone="123-4567"
```

```
name="Evan Weaver"  
echo $name  
Evan Weaver  
name="Murray Saul"  
echo $name  
Murray Saul  
readonly name  
name="Mark Fernandes"  
-bash: name: readonly variable  
  
readonly phone="123-4567"  
phone=456-7891  
-bash: phone: readonly variable
```

INSTRUCTOR DEMONSTRATION

Task1:

Write a Bash shell script to display the following message using an **environment variable** so it will work in any user's terminal if the shell script was issued:

```
My username is: (your-username)
```

Task2:

Write a Bash shell script to prompt the user for their **full name** and prompt the user for their **age** to be stored in **user-defined** variables. Display the following output using the values of those variables:

```
Enter your Full Name: (your full name)
```

```
Enter your Age: (your age)
```

```
Hello, my name is (your full name), and I am (your age) years old.
```



SHELL SCRIPTING

Getting Practice

To get practice to help perform **assignment #3**, perform **Week 10 Tutorial**:

- [INVESTIGATION 1: CREATING A SHELL SCRIPT](#)
- [INVESTIGATION 2: USING VARIABLES IN SHELL SCRIPTS](#)
- [LINUX PRACTICE QUESTIONS](#) (Part A 1 – 2 , Part B **Walk-Thru #1**)