Ifs and Lists

Cybersecurity

Bash Scripting and Programming Day 2

Class Objectives

By the end of today's class, you will be able to:

Read bash and interpret scripts.

Use variables in your bash scripts.

Use if statements in your bash scripts.

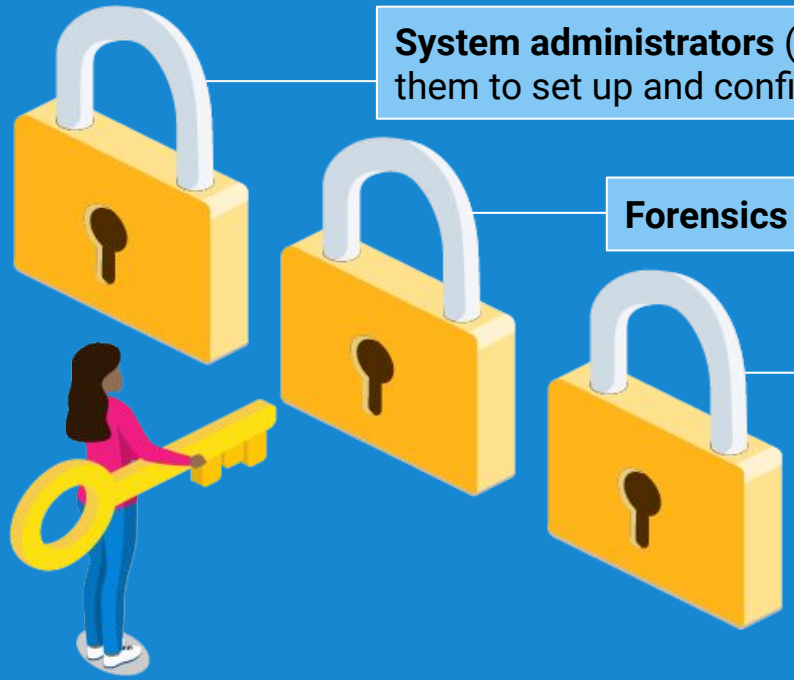
Use lists in your bash scripts.

Iterate through lists with a for loop.

If Statements

Bash Scripting

Bash scripting is a vital skill for any cybersecurity professional. Scripts can be used for the following:



System administrators (sysadmins) use them to set up and configure machines.

Forensics use them for investigations.

Pentesters use them to probe networks and find vulnerabilities.

Conditional Statements

Cyber professional roles often require advanced bash skills. Today, we'll continue to develop our scripting skills in order to incorporate the following into our script:

01

If statements

02

For loops to complete repetitive tasks

03

Automating the setup of a machine

Criteria and Decision Making

First, we will need our scripts to make decisions based on specific criteria.

For example:
Our scripts can check for the existence of users, directories, files, or permissions.

Based on the results, the script can then take specific action or even stop execution.

Criteria and Decision Making

To accomplish these tasks, we will need to learn a few useful scripting techniques:

Using if statements.

Using if/else statements.

Testing multiple conditions using if/else statements.

First, a comment...

```
# This is a comment
```

Comments are non-executable text within the script.

Placing a # in front of a line tells bash to ignore it. This is known as “commented out”.

These comments can describe the script functionality in plain English, making it easier for developers to understand the code.

Commenting out is often preferable to simply deleting the code. It also allows developers to toggle certain code on and off.

If Syntax

```
if [ <test> ]
then
<run_this_command>
fi
```

`if`	Initiates our if statement.
`[]`	Encapsulates the condition.
`then`	Then runs following commands if the condition is met.
`fi`	Ends the if statement.

If Syntax

```
if [ 5 -gt 8 ]
then
    echo “This doesn’t make sense!”
fi
```

`if [ 5 -gt 8 ]`	This will check if 5 is greater than 8.
`then`	Runs the following commands if the condition is met.
`echo “This doesn’t make sense!”`	Will have the script print to the screen “That doesn't make sense”.
`fi`	Ends the if statement.

If Else Syntax

```
if [ <test> ]
then
    <run_this_code>
else
    <run_this_code>
fi
```

`if [ <condition> ]`	If this test is true...
`then`	Runs the following code.
`else`	Runs the following code if the condition is false.
`fi`	Ends the if statement.

If Else Syntax

```
if [ 5 -gt 4 ]
then
   echo "That is correct!"
else
   echo "That doesn't make sense!"
fi
```

`if [ 5 -gt 4 ]`	This will check if 5 is greater than 4.
`then`	Runs following code if the condition is met.
`echo “That is correct!”`	Is printed to the screen.
`else`	Runs the following code if the condition is unmet: **`echo “This doesn’t make sense!”`**
`fi`	Ends the if statement.

&& Syntax

```
if [ <condition_1> ] && [ <condition_2> ]
then
    <run_this_code>
fi
```

`if [<test1>]`	Checks if one condition is true.
`&& [<test2>]`	Checks if a second condition is true.
`then`	Runs the following code if both conditions are met.
`fi`	Ends the if statement.

&& Syntax

```
if [ 5 -gt 4 ] && [ 4 -gt 3 ]
then
    echo “That makes sense”.
fi
```

If the following two conditions are met:

`if [ 5 -gt 4 ]`

`&& [4 -gt 3]`

Run `echo “That makes sense”.`

|| Syntax

```
if [ <condition1> ] || [ <condition2> ]
then
<run_this_code>
else
<run_this_code>
fi
```

`if [ <condition1> ]`	If condition 1 is true.
`|| [<condition 2>]`	Or if condition 2 is true.
`then`	Run the following code.
`else`	Runs code if both conditions are false.
`fi`	Ends the if statement.

|| Syntax

```
if [5 -gt 4] || [4 -gt 3 ]
then
    echo “That only partially makes sense”
else
    echo “None of this makes sense”
fi
```

`if [ 5 -gt 4]`	If this condition is true...
`|| [4 -gt 3]`	...or if this test is true...
`then`	Run the following code.
`else`	Otherwise (if both conditions are false), run the following command.
`fi`	Ends the if statement.

Summary

`if`	`if [ <condition> ]`
	`then`
	`<run_this_command>`
	`fi`
		Runs code *if* the condition is met.

`if / else`	`if [ <condition> ]`
	`then`
	`<run_this_command>`
	`else`
	`<run_this_command>`
	`fi`
		Runs code *if* the condition is met. If condition isn't met, it will run a different command.

`&&`	`if [ <condition1> ] && [ <condition2> ]`
	`then`
	`<run_this_command>`
	`fi`
		Runs code if more than one condition is met.

`||`	`if [ <condition1> ] || [ <condition2> ]`
	`then`
	`<run_this_command>`
	`else`
	`<run_this_command>`
	`fi`
		Runs code if only one of multiple conditions are met.

Creating Conditionals

Variables

A **variable** is a container used to hold a specific value. Different variables can hold different types of data. Common variable types include:

String

A data type composed of a sequence of textual and/or numerical characters.

Examples of string data include:

- Names of people
- Phone numbers
- Words/sentences

Integer

A data type that's a whole number value.

We can perform arithmetic operations on these variables.

Now, we'll compare variables using the following conditionals:

`==`	This string is equal to another.
`!=`	If two strings are not equal.
`-gt`	If one integer is greater than another.
`-lt`	If one integer is less than another.
`-d /path_to/directory`	Checks for existence of a directory.
`-f /path_to/files`	Checks for existence of a file.

Equals to

```
# If $x is equal to $y, run the echo command.
if [ $x = $y ]
then
 echo "X is equal to Y!"
fi
```

`if [ $x = $y ]`	If the value of the x variable is equal to the value of the y variable.
`then`	Then, run the following command.
`echo "X is equal to Y!"`	The echo command that will run if the initial condition is met.
`fi`	Ends the if statement.

Not equals to

```
# If $x is not equal to $y, run the echo command.
if [ $x != $y ]
then
 echo "X does not equal Y!"
fi
```

`if [ $x != $y ]`	If the value of the x variable is not equal to the value of the y variable.
`then`	Then, run the following command.
`echo "X does not equal Y!"`	The echo command that will run if the initial condition is met.
`fi`	Ends the if statement.

Conditionals and Strings

```
# If str1 is not equal to str2, run the echo command and exit the script.
if [ "$str1" != "$str2" ]
then
  echo "These strings do not match."
  echo "Exiting this script."
fi
```

`if [ "$str1" != "$str2" ]`	If string 1 is not equal to string 2...
`then`	Then, run the following command.
`echo “These strings do not match"`	
`echo “Exiting this script.”`	
`fi`	Ends the if statement.

Greater Than and Less Than

```
# If x is greater than y, run the echo command
if [ $x -gt $y ]
then
  echo "$x is greater than $y".
 fi
```

```
# Checks if x is less than y
if [ $x -lt $y ]
then
  echo "$x is less than $y!"
else
  echo "$x is not less than $y!"
 fi
```

Checking Files and directories

```
# check for the /etc directory
if [ -d /etc ]
then
  echo The /etc directory exists!
fi
```

```
if [ -d /etc ]...
```

If the /etc directory exists, run the following echo command.

```
# check for my_cool_folder
if [ ! -d /my_cool_folder ]
then
  echo my_cool_folder is not there!
fi
```

```
if [ ! -d /my_cool_folder ]...
```

If /my_cool_folder does not exist, run the following echo command.

```
# check for my_file.txt
if [ -f /my_file.txt ]
then
  echo my_file.txt is there
fi
```

```
if [ -f /my_file.txt ]...
```

If the file /my_file.txt exists, then run the following echo command.

We can use built-in variables and command expansions inside our tests.

Built In Variables and Command Expansions

In the following three examples, we will use these variables and command expansions to verify:

- If certain users are sysadmin.
- The UID of the users.
- The current user running the script is a sysadmin.

Built In Variables and Command Expansions

```
# if the user that ran this script is not the sysadmin user, run the echo command
if [ $USER != 'sysadmin' ]
then
  echo “You are not the sysadmin!”
  exit
fi
```

```
if [ $USER != 'sysadmin' ]
```

If the user that ran this script is not “sysadmin,” then run the following echo command.

Built In Variables and Command Expansions

```
# if the uid of the user that ran this script does not equal 1000, run the echo command
if [ $UID != 1000 ]
then
  echo your UID is wrong
  exit
fi
```

```
if [ $UID != 1000 ]
```

If the UID of the user who is running this script does not equal 1000, then run the following echo command.

Built In Variables and Command Expansions

```
# if the sysadmin ran the script, run the echo command
if [ $(whoami) = 'sysadmin']
then
  echo 'you are the sysadmin!'
fi
```

```
if [ $(whoami) = 'sysadmin']
```

If the user “sysadmin” ran the script, then run the following echo command.

Activity: Variables and If Statements

In this activity, you will add variables and conditional if statements to your scripts.

Suggested Time:

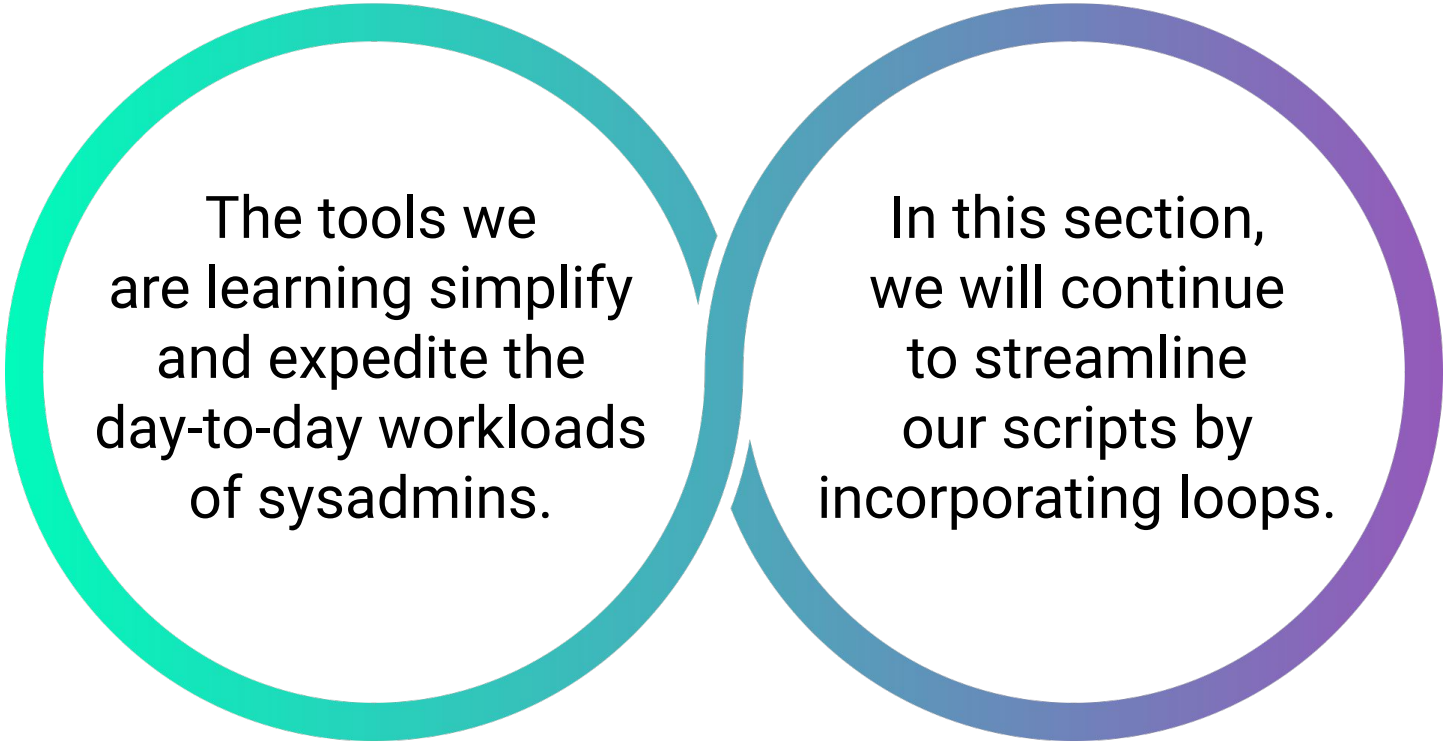
20 Minutes

Time’s Up! Let’s Review.

Questions?

List and Loops

Optimizing our Scripts



Sysadmins will often complete the same common task over and over and over again.

For loops allow us to run a block of code multiple times in a row, without having to repeatedly enter that code.

For Loops

The code is run against each item in a list of items. The for loop runs for as many times as there are items on the loop.

```
for package in ${packages[@]}
do
  if [ ! $(which $package) ]
  then
    apt install -y $package
  fi
    done
```

Instructor Demonstration

For Loops

Demo Summary

Made lists

```
my_list=(a b c d e f)
```

Created for loops

```
# for <item> in <list>;
# do
#   <run_this_command>
#   <run_this_command>
# done
```

Accessed the list with commands

```
$ echo ${my_list[0]}
a
$ echo ${my_list[4]}
e
$ echo ${my_list[@]}
  a b c d e f
```

Created loops with conditionals

```
# run an operation on each number
for num in {0..5};
do
    if [ $num = 1 ] || [ $num = 4 ]
    then
      echo $num
    fi
  done
```

Activity: Lists and Loops

In this activity, you use for loops to automate repetitive tasks.

Suggested Time:

20 Minutes

Time’s Up! Let’s Review.

Questions?

Countdown timer

15:00

(with alarm)

For Loops for Sysadmins

Sometimes, creating a script might be excessive for the scale of the present task. Instead, we can use these for loops directly on the command line, outside of a script.

For Loops for SysAdmins

For example, suppose that a sysadmin needs to prep each employees computer with the following packages:

Rather than just logging onto each of the computers and installing the packages, the sysadmin could turn these apps into a list.

```
packages = [Zoom, Slack, Excel, Docusign]
```

We can then employ a for loop to iterate through the packages list and install the apps on each machine.

Why For Loops

Loop through...

...a list of packages to check if they are installed.

...the results of a find command and take action on each item found.

...a group of files, check their permissions, and change if needed.

...a group of files and create a cryptographic hash of each file.

...all the users on the system and take an action for each one.

For Loops for Sysadmins

In the next demo, you will see how loops can be used to:

- Run through a list of packages and check if certain ones are installed.
- Search users' home directories for scripts and print a confirmation statement.
- Loop through scripts in your scripts folder and change the permissions to execute.
- Create a `for` loop that moves through a group of files and creates a hash of each file.

We will also write for loops directly on the command line.

Instructor Demonstration

Scripts and Loops

Activity: For Loops for SysAdmins

In this activity, you will add loops to your `sys_info.sh` script, and run loops directly on the command line.

Suggested Time:

25 Minutes

Time's Up! Let's Review.

Questions?

Optional:
Script Along Set Up

- So far, we have used if statements to control the execution flow in a script.
- We also used if statements with loops to perform administrative tasks.
- For our final activity, we apply these new skills by going through a completed user setup script.

Instructor Demonstration

Script Along


The End