Ifs and Lists

Cybersecurity

Bash Scripting and Programming Day 2

© 2019 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

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

Scripting plays an important in many cybersecurity roles:

- Sysadmins use them to setup and configure machines.
- Forensics use them for investigations.
- Pen testers use them to probe networks and find vulnerabilities.

Conditional Statements

Those roles and responsibilities require advanced bashing skills.Today, we’ll continue to develop our scripting skills in order to incorporate the following into our script:

- If statements and the decision making based on criteria.
- For loops to complete repetitive tasks.
- Automating the set up 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 users, directories, files or permissions. Based on the results, the script can then take specific action or even stop its execution.

In order to accomplish this feature, we will need to learn a few useful scripting techniques:

- Using if statements.
- Using if/else statements.
- Comparing multiple tests 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.
- These comments can help inform users what the script is supposed to do.
- When lines are ignored, they are **commented out.**

If Syntax

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

`if:` initiates our if statement.

`[]:` encapsulates the condition.

`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 to see 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_command>
else
   <run_this_command>
fi
```

`if [ <test> ]:` if this test is true...

`then:` run the following commands.

`else:` if `[ <test> ]` is false, the run the following command.

`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 to see if 5 is greater than 4.

`then`: Runs following commands *if* the condition is met.

`echo “That is correct!”` is printed to the screen.

`else:` Run the following if the condition is unmet

`echo “This doesn’t make sense!”`

`fi`: Ends the if statement.

&& Syntax

```
if [ <test1> ] && [ <test2> ]
then
<run_this_command>
fi
```

`if [ <test1> ]:` If this test is true...

`&& [<test2>]:` ...and if this is true,

`then:` Run the following commands.

`fi:` Ends the if statement.

&& Syntax

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

`if [5 -gt 4]:` Will test the first argument.

`&& [4 -gt 3]`: If the first argument *and* the second argument are true, run the following command.

|| Syntax

```
if [ <test1> ] || [ <test2> ]
then
<run_this_command>
else
<run_this_command>
fi
```

`if [ <test1> ]:` If this test is true...

`|| [<test2>]:` ...or if this test is true...

`then:` Run the following commands.

`else:` Otherwise (if both conditions are false), run the following command.

`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 test is true...

`|| [4 -gt 3]:` ...or if this test is true...

`then:` Run the following commands.

`else:` Otherwise (if both conditions are false), run the following command.

`fi:` Ends the if statement.

Summary:

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

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

&&	`if [ <test1> ] && [ <test2> ]`
	`then`
	`<run_this_command>`
	`fi`
		Runs a command if more than one condition need to be met.

||	`if [ <test1> ] || [ <test2> ]`
	`then`
	`<run_this_command>`
	`else`
	`<run_this_command>`
	`fi`
		Runs command only one of multiple conditions needs to be met.

Creating Conditionals

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

`=`	If two items are equal.
`!=`	If two items 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.
`-e /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!"`

`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

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

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

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

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

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

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

`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.

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

- Specific users are sysadmin users;
- The UID of a user is specific to a certain value;
- The sysadmin is the user that is running the current script.

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 the sysadmin user, 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 -ne 1000 ]
then
  echo your UID is wrong
  exit
fi
```

`if [ $UID -ne 1000 ]`: If the UID of the user that ran 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 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:
10 minutes

Activity Review: Variables and If Statements

Create a variable to hold the path of your output file.	
Create a variable to hold the output of the command: `ip addr | grep inet | tail -2 | head -1`	
Create a variable to hold the command: `find / -type f -perm /4000`	

Activity Review: Variables and If Statements

Create a variable to hold the path of your output file.	`output =$HOME/research/sys_info.txt`
Create a variable to hold the output of the command: `ip addr | grep inet | tail -2 | head -1`	`ip=$(ip addr | head -9 | tail -1)`
Create a variable to hold the command: `find / -type f -perm /4000`	`suids=$(find / -type f -perm /4000)`

Activity Review: Variables and If Statements

Create an if statement that checks for the existence of the ~research directory. If the directory exists, do nothing. If the directory does not exist, create it.	
Create an if statement that checks for the existence of the file ~/research/sys_info.txt	

Activity Review: Variables and If Statements

Create an if statement that checks for the existence of the ~research directory. If the directory exists, do nothing. If the directory does not exist, create it.	`#remove`
	`mkdir ~/research 2> /dev/null`
	
	`#replace it with`
	`if [ ! -d $HOME/research ]`
	`then`
	`mkdir $HOME/research`
	`fi`

Create an if statement that checks for the existence of the file ~/research/sys_info.txt	`if [ -f $output ]`
	`then`
	` rm $output`
	`fi`

Activity Review: Variables and If Statements

Create an if statement that checks to see if the script was run using sudo.	
Other ways to write this statement:	

Activity Review (Bonus): Variables and If Statements

Create an if statement that checks to see if the script was run using sudo.

```
if [ $UID -eq 0 ]
then
 echo “Please do not run this script with sudo.”
 exit
fi
```

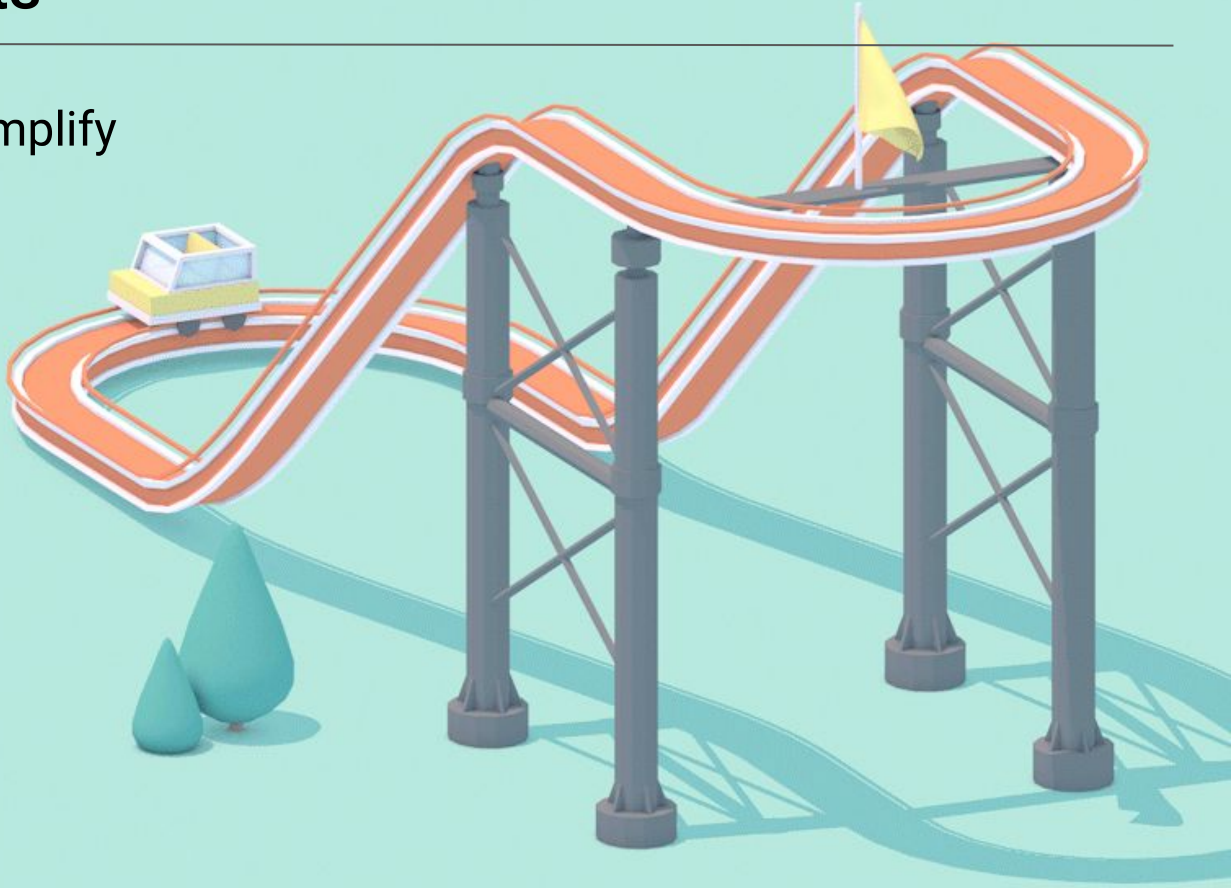
Other ways to write this statement:

```
# Check the contents of the $USER variable against root.
if [ $USER = `root` ]

#Check the output of whoami command against root.
if
```

List and Loops

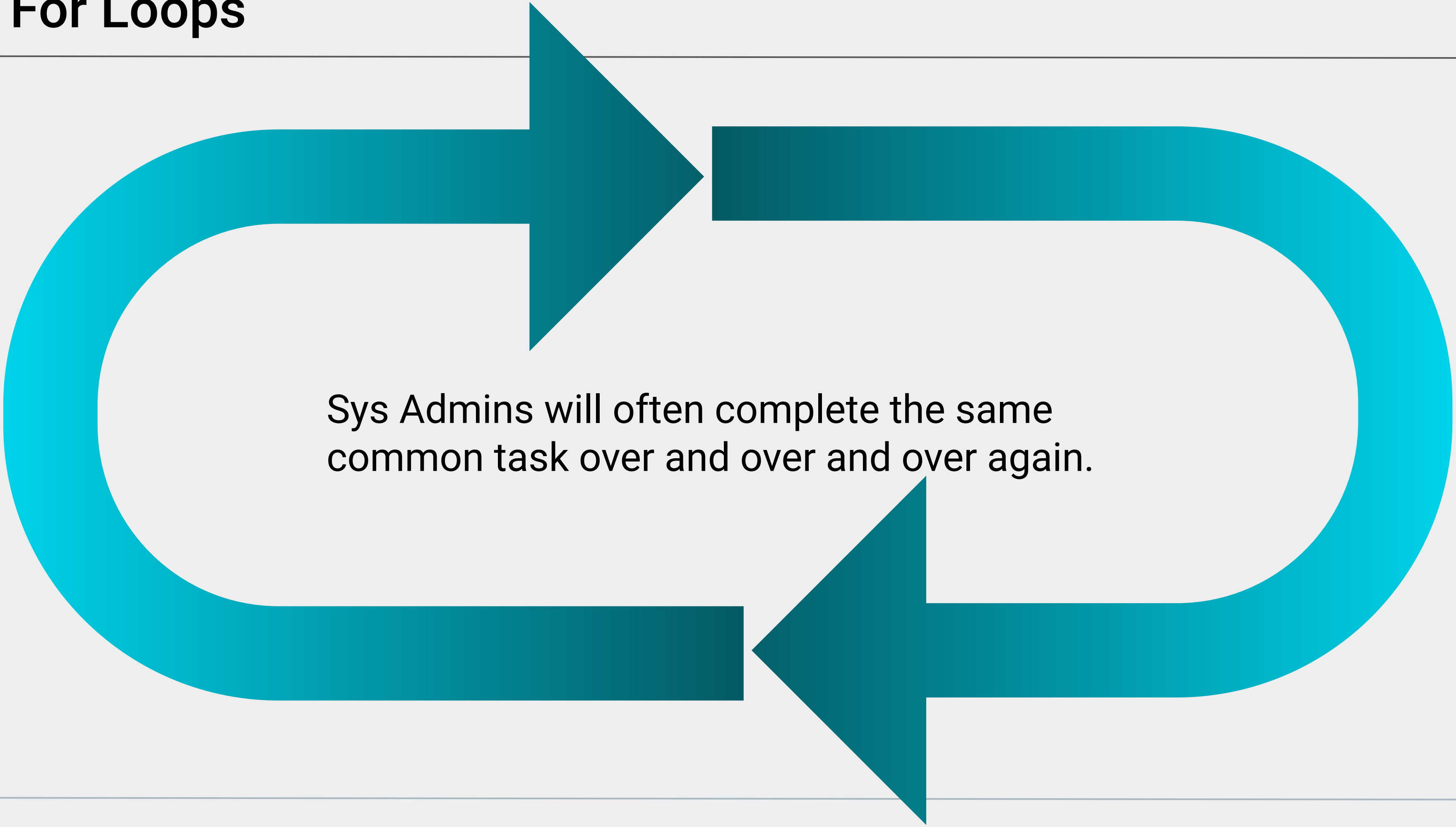
Optimizing our Scripts



The tools we are learning simplify and expedite the day-to-day workloads of sysadmins.

In this section, we will continue to streamline out scripts by incorporating iterative commands known as loops.

For Loops



Sys Admins will often complete the same common task over and over and over again.

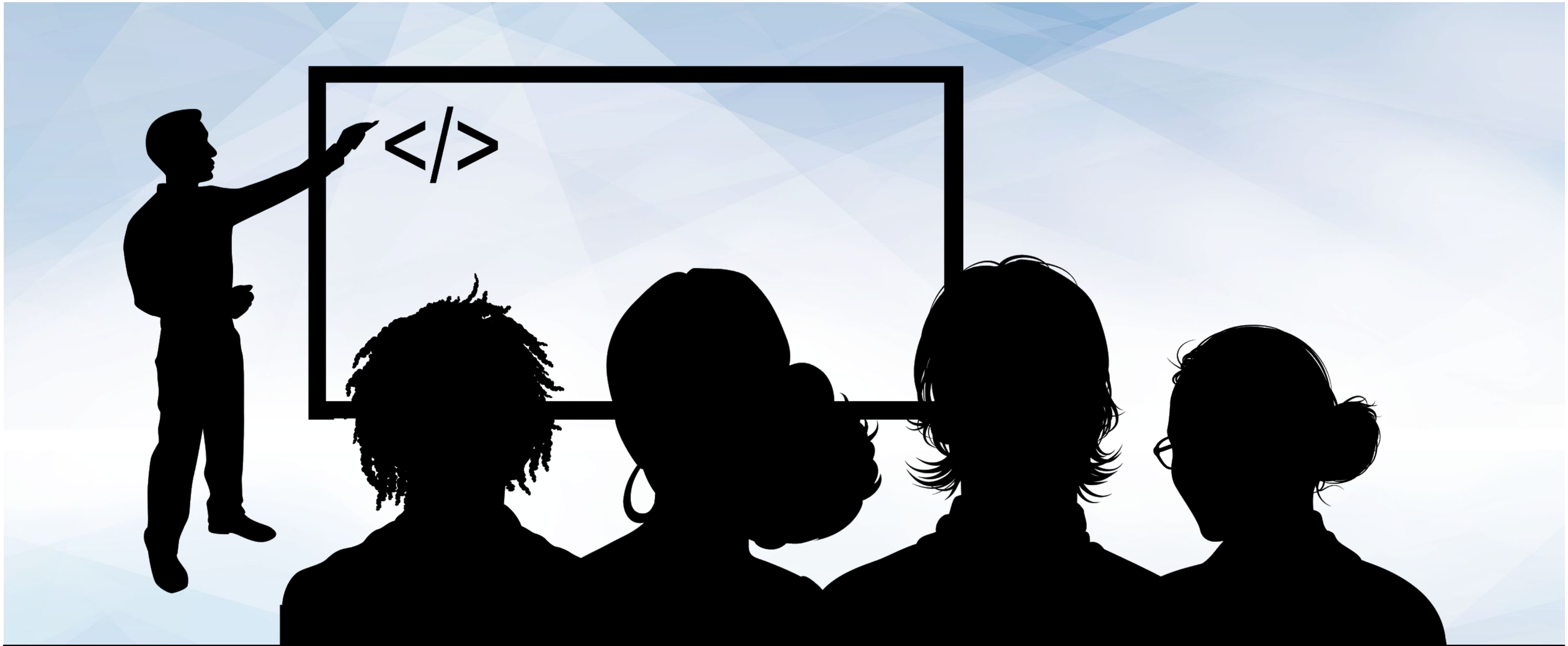
For Loops

For loop allows us to run a block of code multiple times in a row, without having to repeatedly type out that code.

The code is run against a list that holds the items in the code block.

The `for loop` run 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

In the previous demo, we:

Made lists

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

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 for loops

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

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 will start using loops to automate repetitive tasks. You will create several `for` loops that satisfy given requirements. If you get to the bonus task, you can incorporate a for loop into your script.

Suggested Time:
18 minutes

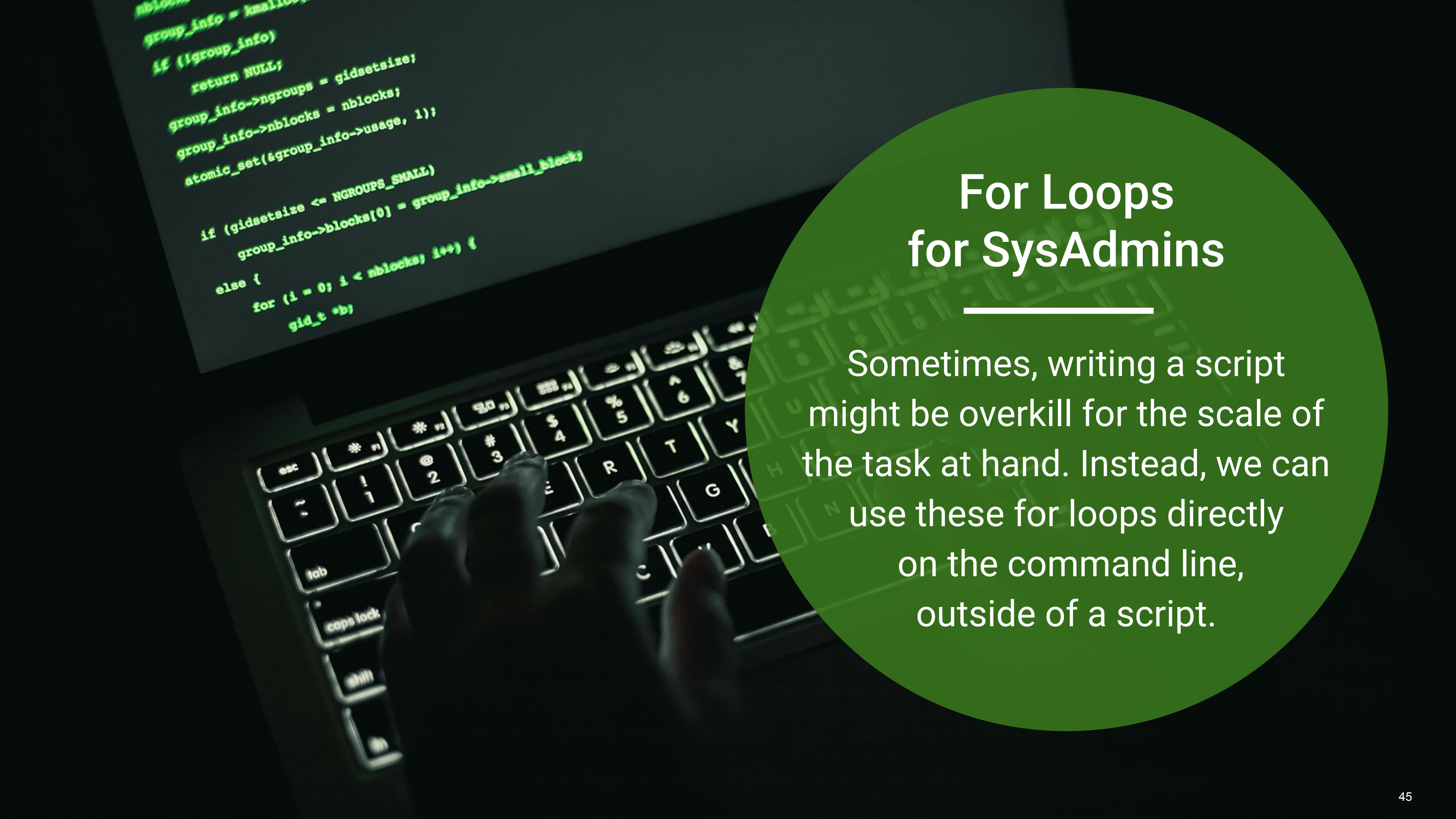
Times Up! Let’s Review.

Countdown timer

15:00

(with alarm)

For Loops for SysAdmins



```
group_info = kmalloc
if (!group_info)
    return NULL;
group_info->ngroups = gidsetsize;
group_info->nblocks = nblocks;
atomic_set(&group_info->usage, 1);

if (gidsetsize <= NGROUPS_SMALL)
    group_info->blocks[0] = group_info->small_block;
else {
    for (i = 0; i < nblocks; i++) {
        gid_t *b;
```

For Loops for SysAdmins

Sometimes, writing a script might be overkill for the scale of the task at hand. Instead, we can use these for loops directly on the command line, outside of a script.

Why For Loops

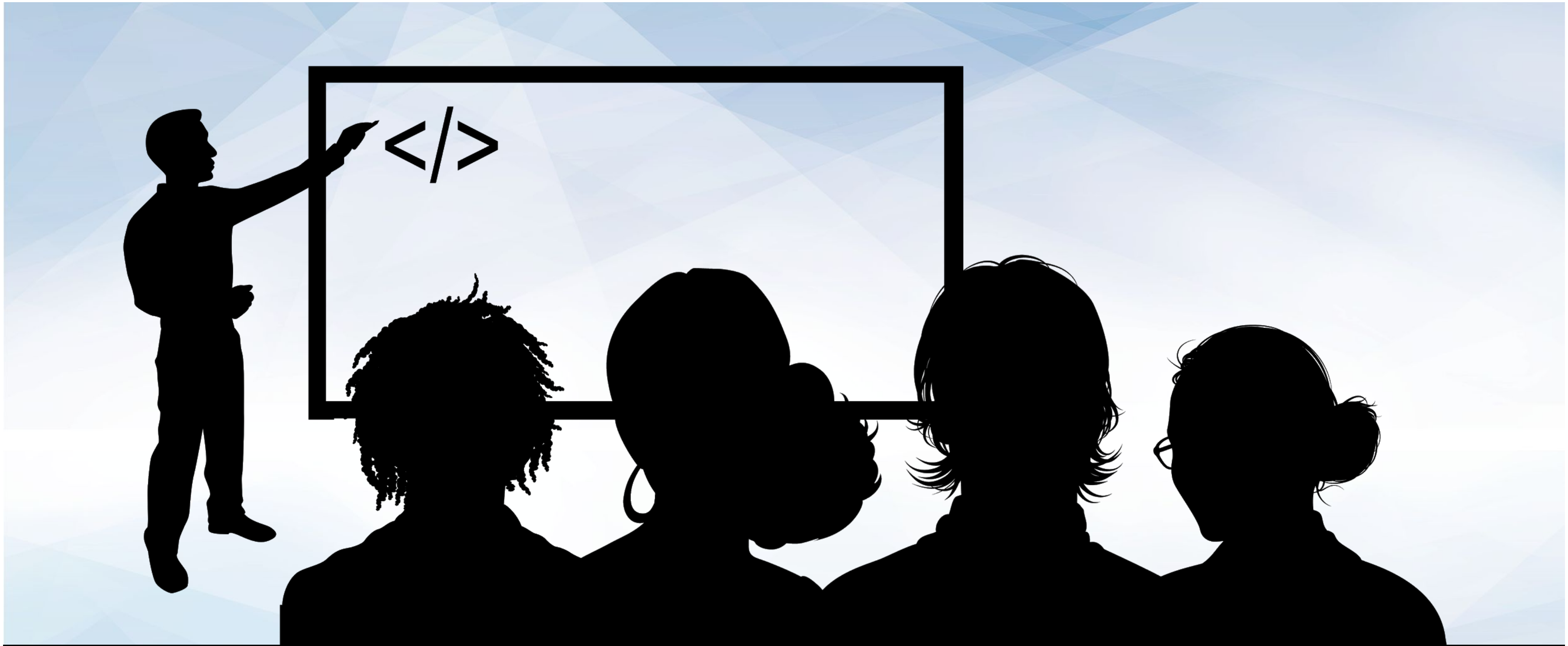
- Loop through a list of packages to check if they are installed.
- Loop through the results of a find command and take action on each item found.
- Loop through a group of files, check their permissions and change if needed.
- Loop through a group of files and create a cryptographic hash of each file.
- Loop through 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 directory 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 use the `for` loops you created to complete relevant sysadmin. You will work in pairs to take a look at a few useful loops you can add to your `sys_info.sh` script, as well as loops you can use directly in the command line.

Suggested Time:
15 minutes

Times Up! Let’s Review.

Activity Review: For Loops for SysAdmins

Create a list of the paths of sensitive files in the /etc directory that you would like to check the permissions of. This list should have at least the shadow, passwd and hosts files.

```
files=(
'/etc/passwd'
'/etc/shadow'
'/etc/hosts'
)
```

Create a for loop that prints outs the permissions of each file in your file list.

```
for file in ${files[@]};
do
  ls -l $file >> $output
done
```

Add comments into our script.

```
#Display CPU usage
echo -e "\nCPU Info:" >> $output
lscpu | grep CPU >> $output

# Display Disk usage
echo -e "\nDisk Usage:" >> $output
df -H | head -2 >> $output
```

Activity Review: For Loops for SysAdmins

Bonus 1: Create a `for` loop that checks the `sudo` abilities of each user that has a home folder on the system.

```
for user in $(ls /home);
do
  sudo -lU $user
done
```

Run directly in your command line

```
for user in $(ls /home); do sudo -lU $user
```

Activity Review: For Loops for SysAdmins

Create a list that contains the commands `date`, `uname -a`, and `hostname -s`.	`commands=( 'date' 'uname -a' 'hostname -s' )`
Remove the lines that use these commands and replace them with a `for` loop that prints out "The results of the ________ command are:" along with the results of running the command.	see code below

```
commands=(
  'date'
  'uname -a'
  'hostname -s'
  )
```

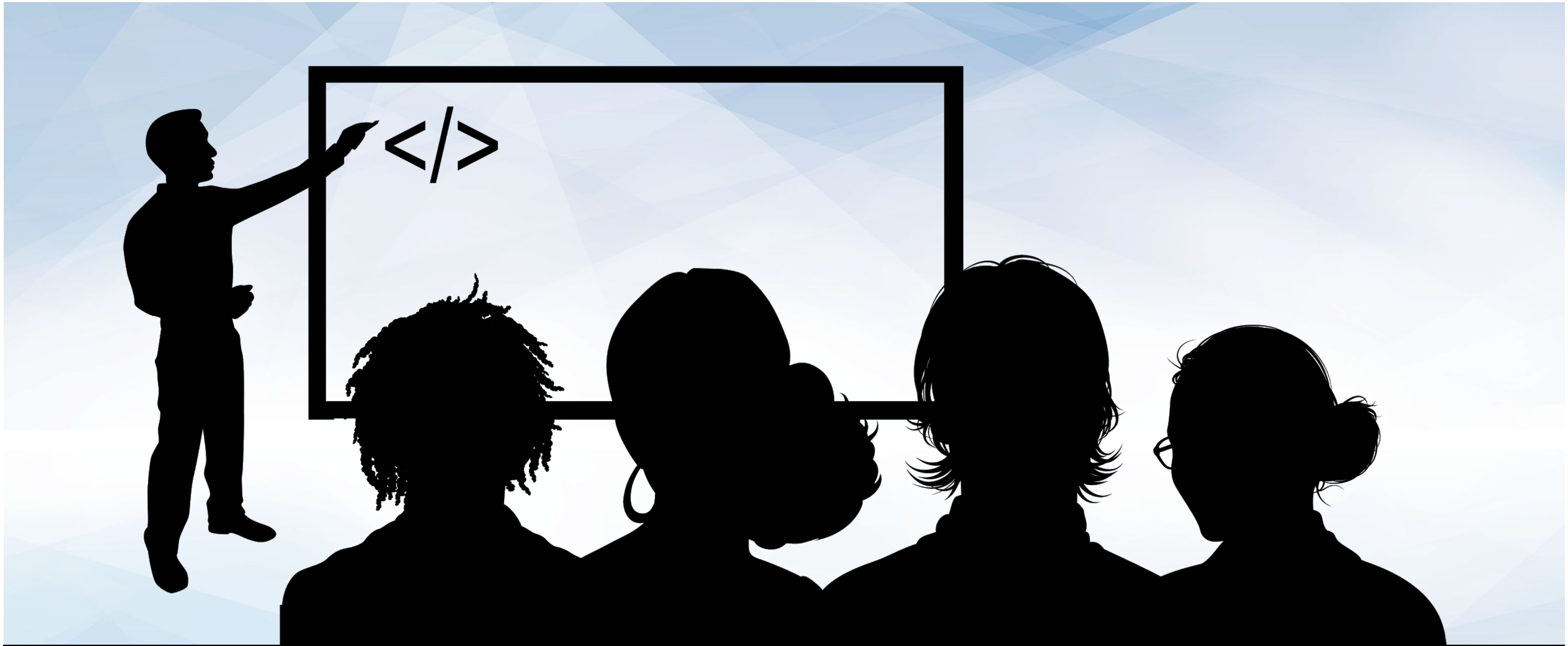
```
for x in {0..2};
do
  results=$(${commands[$x]})
  echo 'Results of "${commands[$x]}" command:' >> $output
  echo $results >> $output
  echo "" >> $output
  done
```

Script Along Set Up

Today in class, we…

- Started with using *if* statements to make decisions in a script.
- Then we moved to using `for` loops and combining `for` loops and `if` statements into loops that accomplish meaningful tasks.

For the rest of class, we will walk through creating a 'setup' script.



Script Along