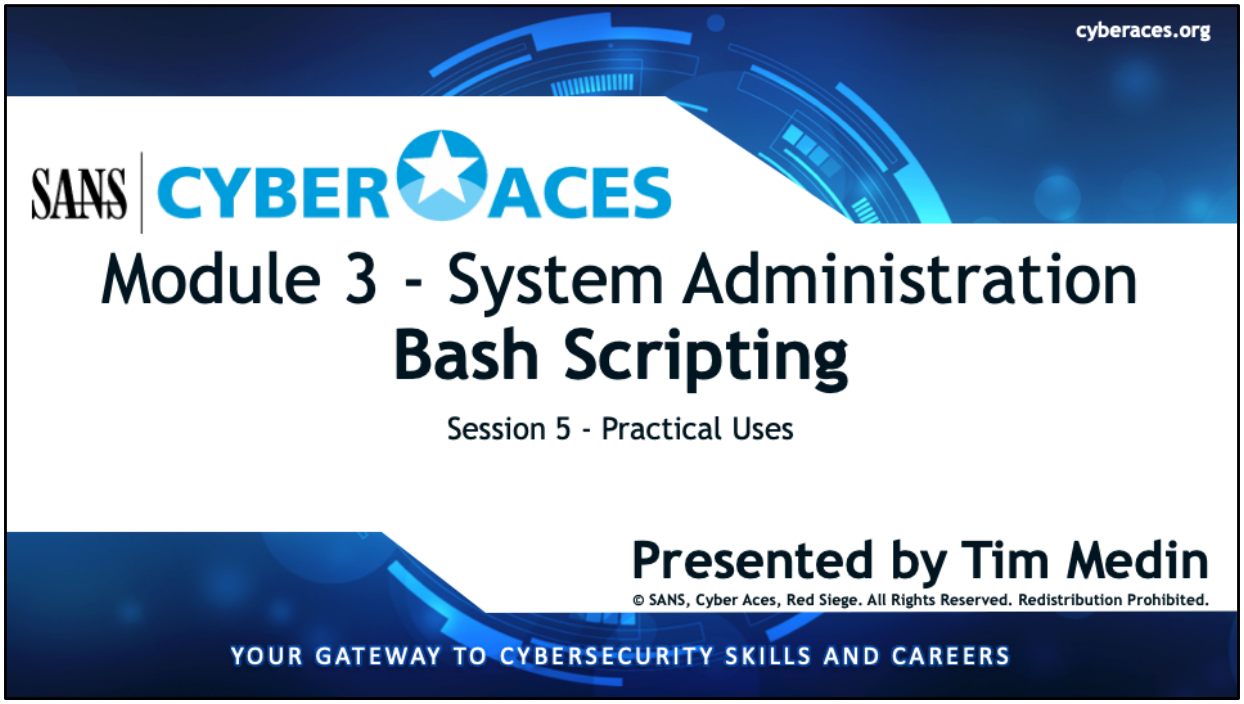
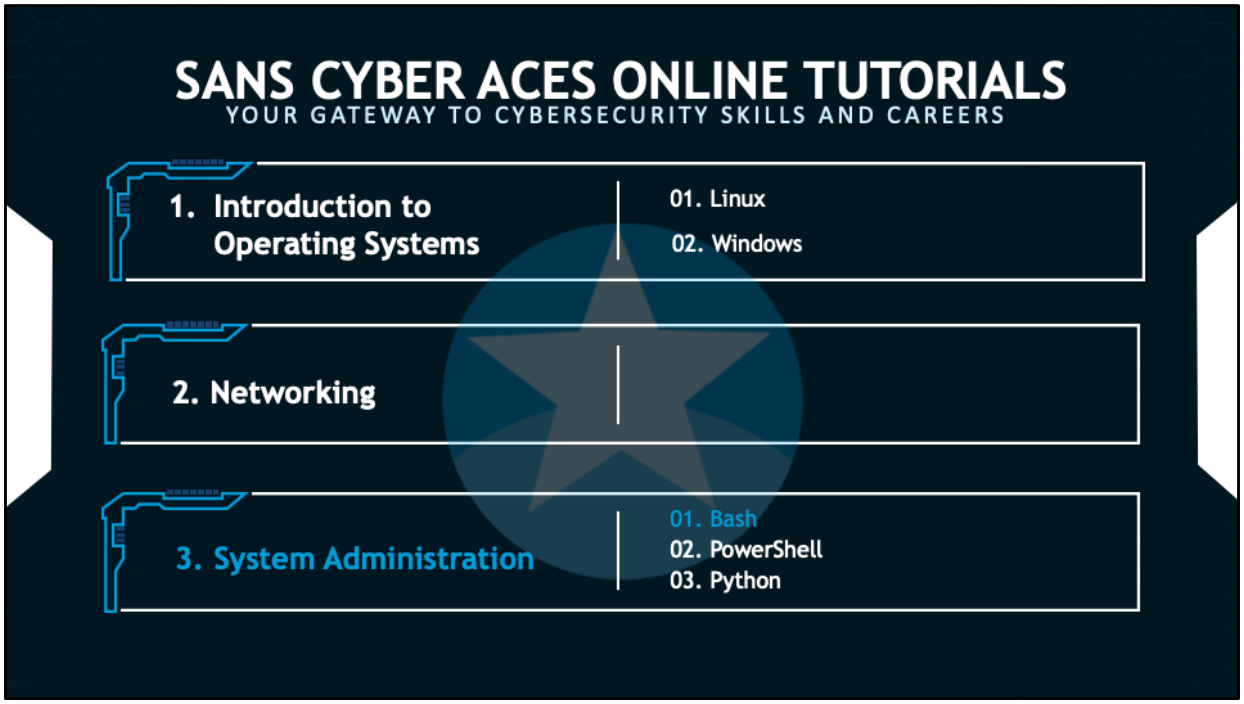


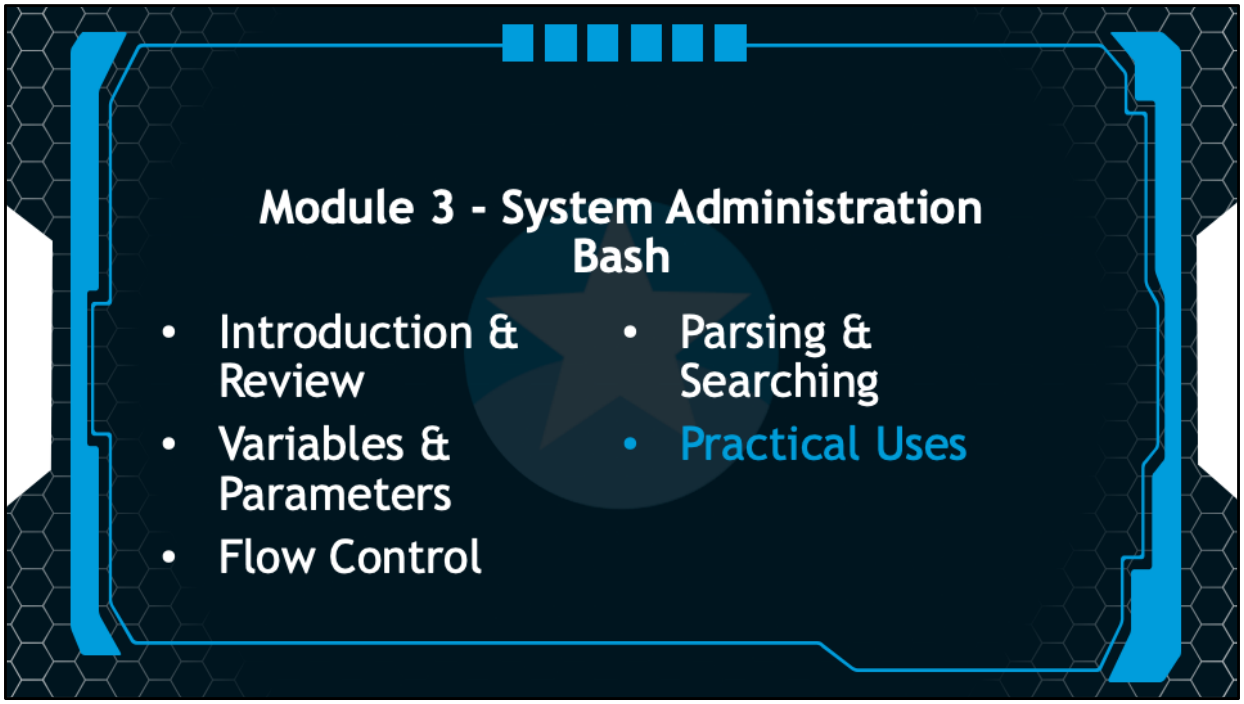
Welcome to Cyber Aces, Module 3! This module provides an introduction to Bash Scripting. In this module we'll use the knowledge we've gained in some practical examples.



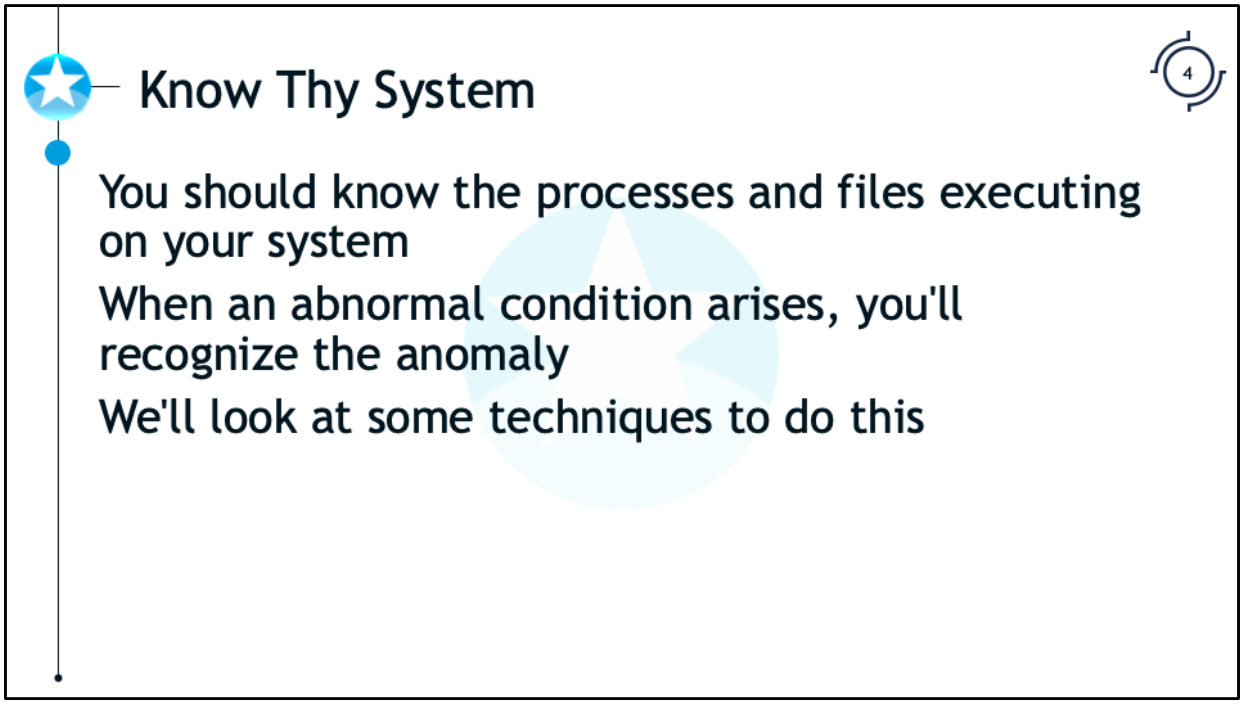
This training material was originally developed to help students, teachers, and mentors prepare for the Cyber Aces Online Competition. This module focuses on the basics of system administration and scripting. . This session is part of Module 3, System Administration. This module is split into three sections, Bash, PowerShell, and Python. In this session, we will continue our examination of Bash.

The three modules of Cyber Aces Online are Operating Systems, Networking, and System Administration.

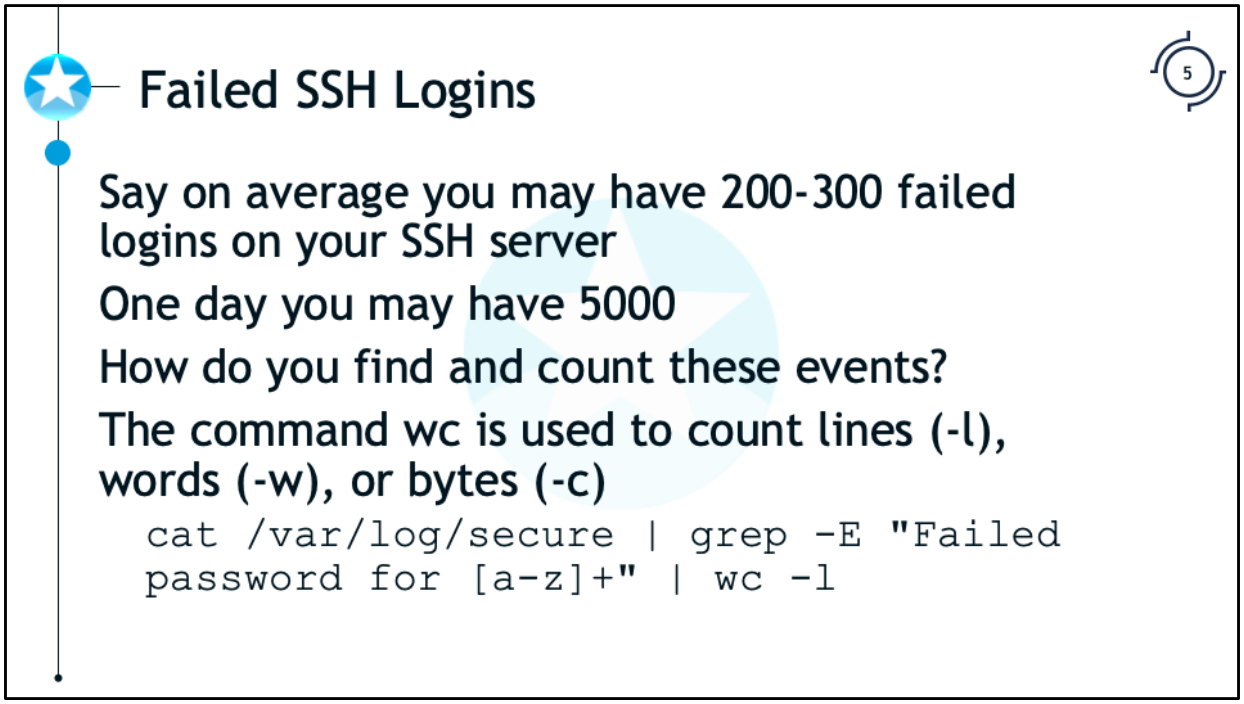
For more information about the Cyber Aces program, please visit the Cyber Aces website at https://CyberAces.org/.



Is this section we will go through some practical scripts.



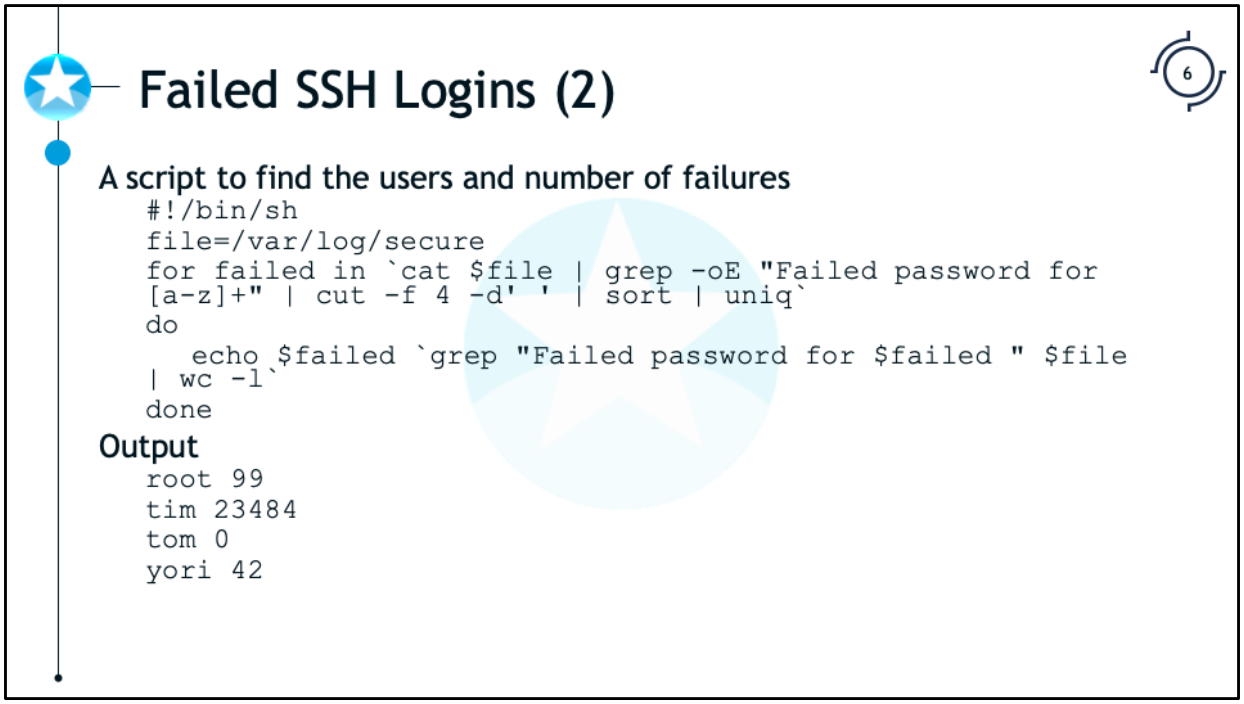
One of the cardinal rules of information security is to "know thy system". This means that you know the processes and files that are executing or stored on your system in a normal state. Then, when an abnormal condition arises, you will recognize the anomaly. Knowing thy system also means that you monitor the logs on your system. In this section, we will look at useful techniques when processing large numbers of log files.



One of the command ways to discover attacks and compromise is to look for things that are out of the ordinary. Looking at SSH logs and seeing an increased number of failed login attempts can show that the system is under attack and could potentially lead you to compromised systems if the login attempts are coming from internal systems.

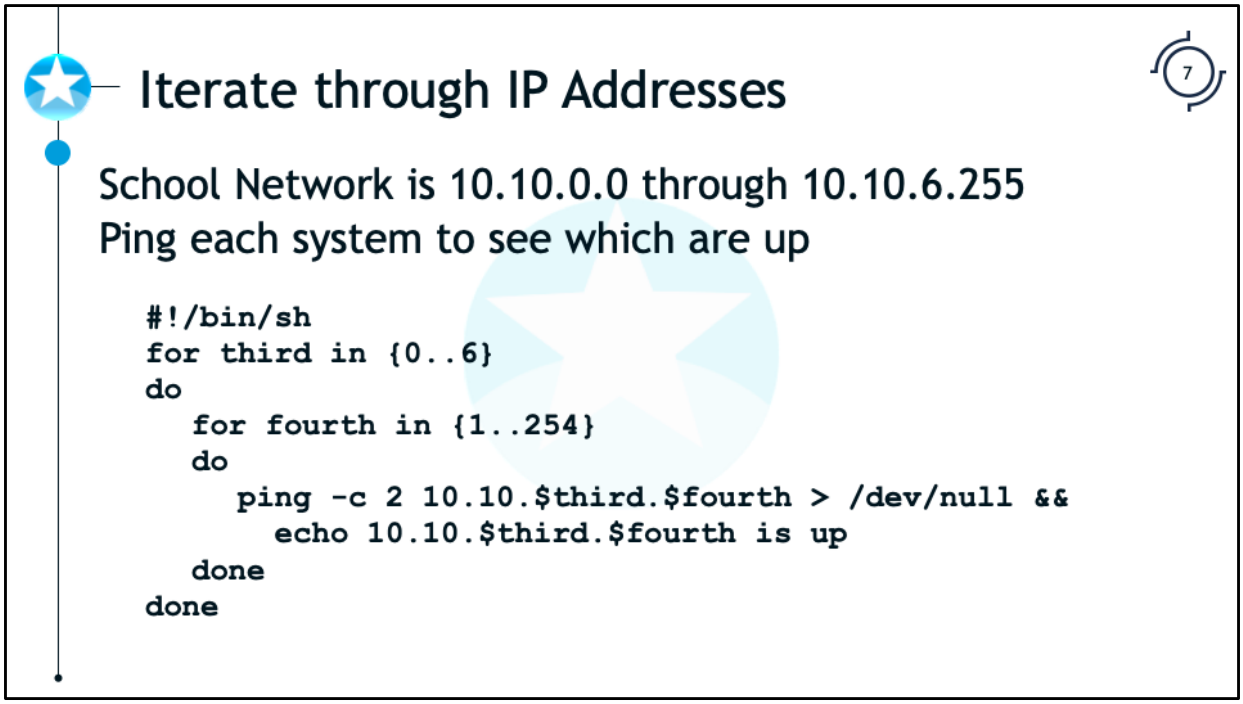
```
$ cat /var/log/secure | grep -E "Failed password for [a-z]+" | wc -l
```

This command will output the file /var/log/secure and pipe it into grep. The grep command will search for "Failed password for " followed by at least 1 alphabet character. The results are piped into wc with the -l option to count the number of lines of output.



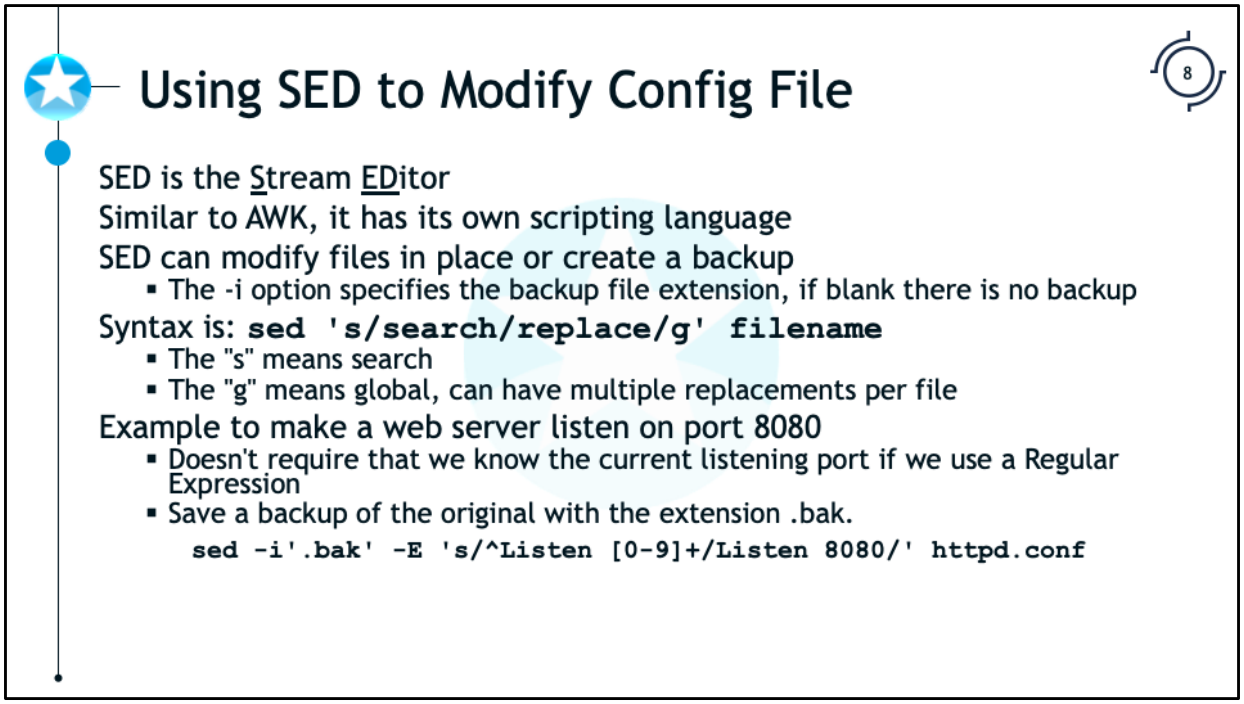
The script explained:

```
#!/bin/bash
# Above is the standard first line of a script
# The file we will be working on
file=/var/log/secure
# The for loop will operate on each line of output using the variable $fail.
# The output is from:
#   - cat the file
#   - search the file using a regex search (-E) for "Failed password for "
#     followed by alpha characters and only return the match not the full line
#     of text (-o)
#   - sort the output and get unique lines
# Each time through the loop the $fail variable will contain something
# similar to "Failed password for root"
for failed in `cat $file | grep -oE "Failed password for [a-z]+" | cut -f4 -d' ' | sort
| uniq`
do
        # Output search string, then the output of the next command (in backticks)
        # The grep command will search the file again for lines matching the
        # current failed user and return the number of lines matched (wc -l). The
        # space is added after $failed so that users tim won't match tim and timmy
        echo $failed `grep "Failed password for $failed " $file | wc -l`
# all done with the For loop
done
```



This example uses nested loops, or one loop inside another loop. The third octet is a number between 0 and 6 and the fourth is between 1 and 254 (Note: we are intentionally skipping 0 and 255).

The ping command will ping the address twice (-c 2) and all output is discarded (redirected to /dev/null). If the ping is successful (has at least 1 successful response) the Echo command will execute and display the IP address that is up.



SED is a handy command used to modify files or output. It is an extremely powerful tool and even has its own scripting language.

One of the nicest features of SED is its ability to modify a file and create a backup. This is a useful safety tool in case the search and replace is typed incorrectly. If you do not want to create a backup you can use -i'' (that's two single quotes not a single double quote) to specify no backup extension.

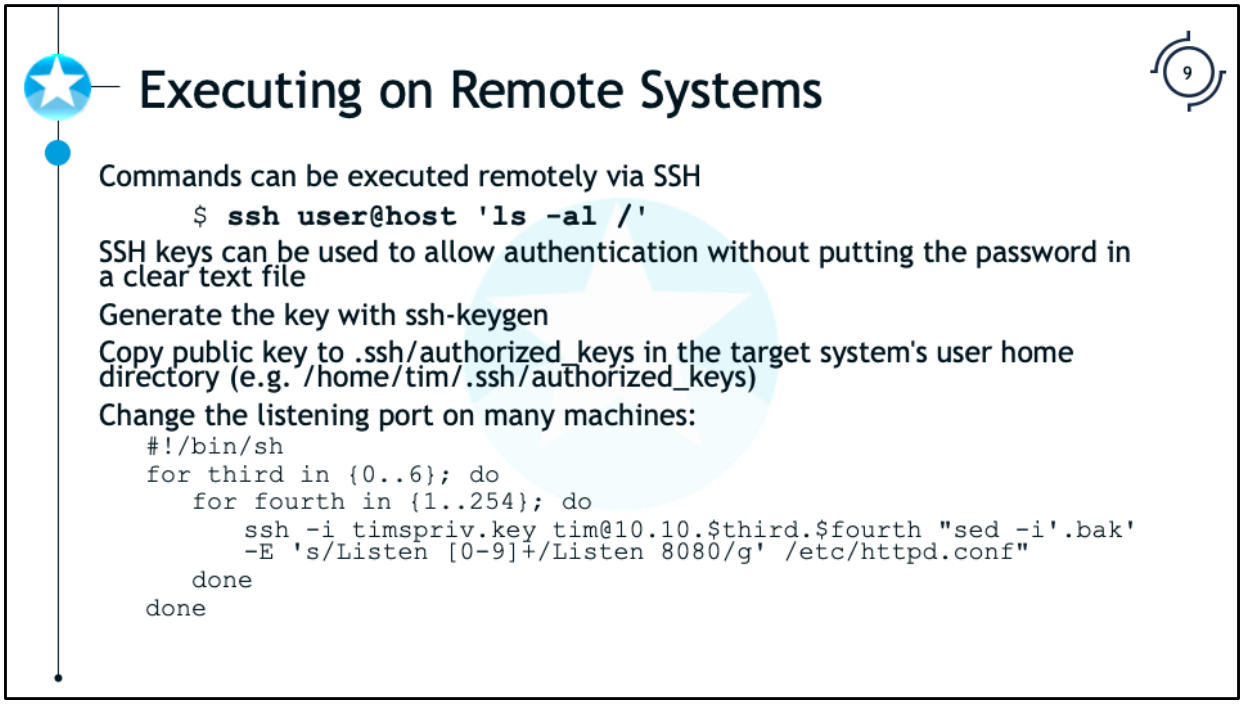
If we want to make a webserver listen on another port we can use sed to find the "Listen" line in the configuration file and replace it. If we use a regular expression we don't need to know the original listening port.

Command:

```
sed -i'.bak' -E 's/Listen [0-9]+/Listen 8080/' httpd.conf
```

Command Explained:

-i'.bak'	create a backup file with the extension .bak	
-E	is the regular expression to use for our search and replace	
s	perform a search	
	/^Listen [0-9]+/	Look for lines starting with the word Listen, a space, followed by one or more digits
	/Listen 8080/	Replace with the literal string "Listen 8080"
httpd.conf	The file to modify	



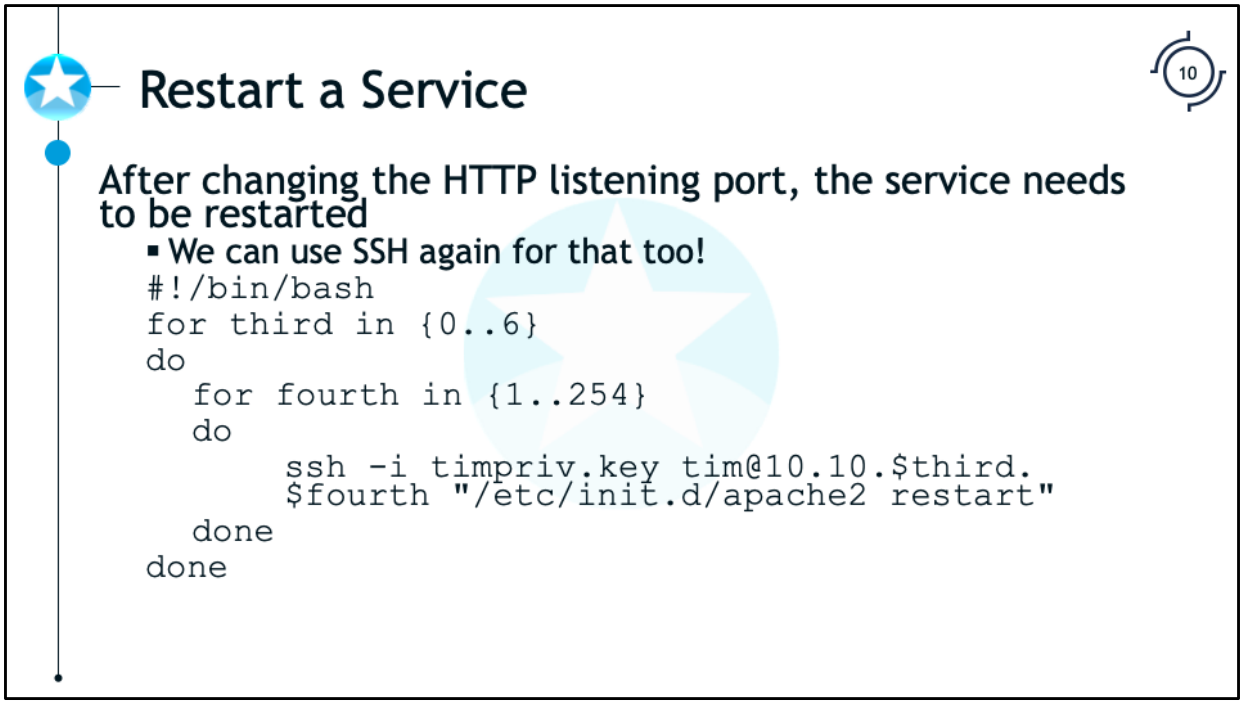
We can execute this remotely on a machine with SSH. If you pass a command to SSH, it will execute that command on the remote host after it logs in. For example:

```
$ ssh username@host 'ls -al'
```

This command will login to 'host' with the username 'username' and will execute the command "ls -la" on the remote host. But what about entering the password? You really don't want to put your password in clear text into a script file. Instead, you would configure SSH private keys so that you can login remotely using the key pairs rather than a password.

Now, combining this with our script above that goes through all of the computers, we come up with the following:

```
#!/bin/bash
# Simple Counting Loops
for thirdoctet in {0..6}
do
    for lastoctet in {1..254}
    do
        ssh -i timpriv.key tim@192.168.$thirdoctet.$lastoctet
        "sed -i'.bak' -E 's/Listen [0-9]+/Listen 8080/g'
        /etc/httpd.conf"
    done
done
```

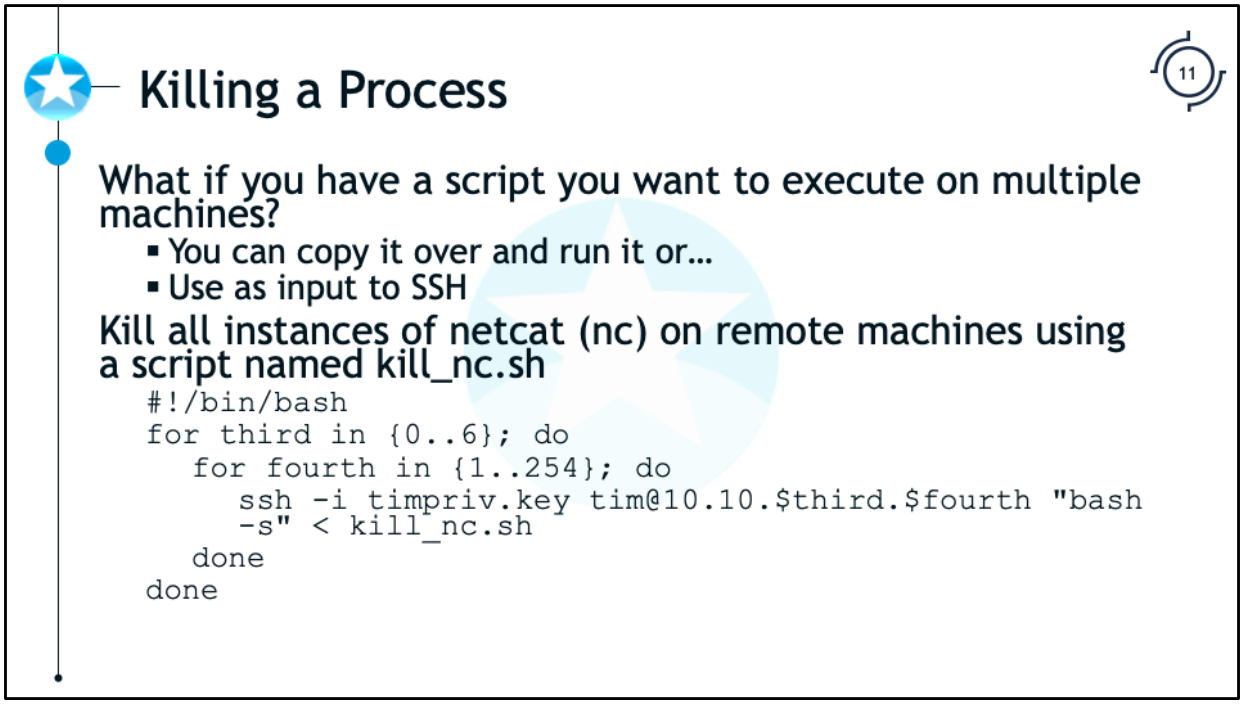


As a system administrator, you may want to restart a service on multiple computers at one time. Our sample script above can easily be modified to accomplish this. The init.d system is used to start and stop services on many Linux distributions. To restart the HTTPD service on all of the computers in our IP range, we could modify our script as follows:

```
#!/bin/bash
#Simple Counting Loops
for third in {0..6}
do
   for fourth in {1..254}
   do
      ssh -i timpriv.key tim@10.10.$third.$fourth "sed    -
      i'.bak' -E 's/Listen [0-9]+/Listen 8080/g' httpd.conf &&
      /etc/init.d/apache2 restart"
   done
done
```

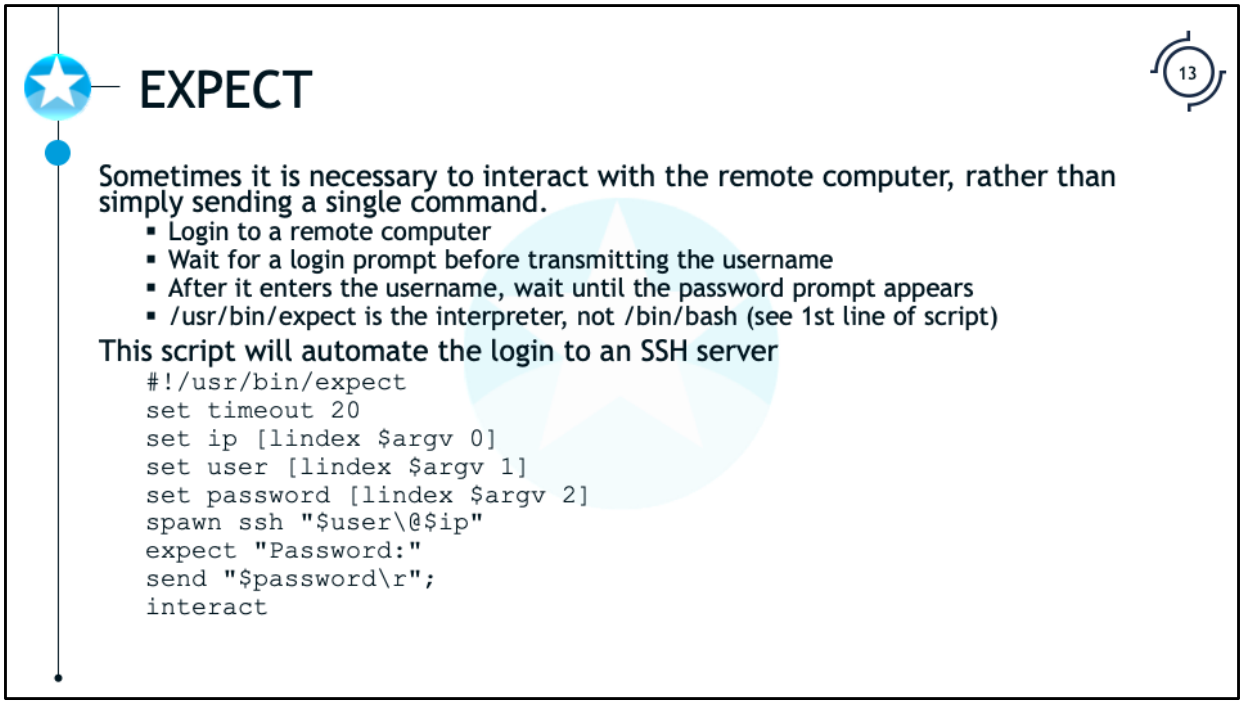
We could even combine the two commands with an && and have it modify the config file and then restart the service if the modification is successful.

```
#!/bin/bash
#Simple Counting Loops
for third in {0..6}
do
   for fourth in {1..254}
   do
      ssh tim@10.10.$third.$fourth "sed -i'.bak' -E 's/Listen
      [0-9]+/Listen 8080/g' httpd.conf && /etc/init.d/apache2
      restart"
   done
done
```



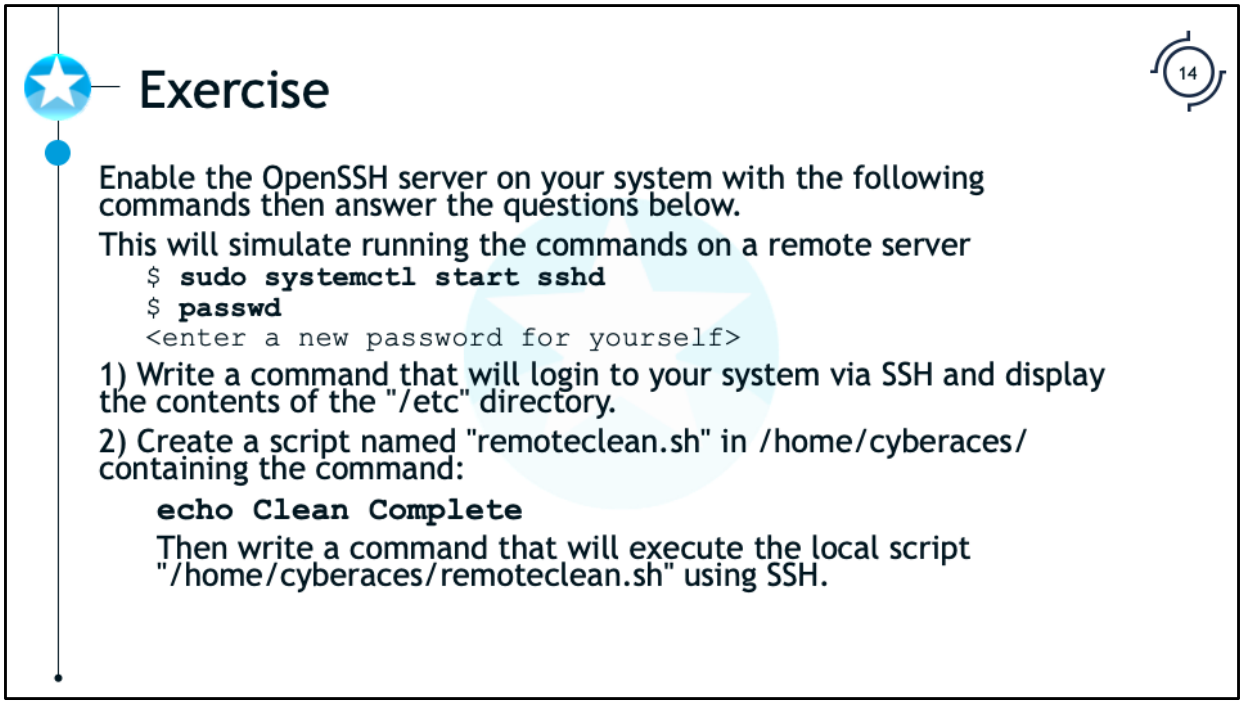
Now we will look at how to do something a little more complex. Imagine that we want to run multiple commands and process the results. SSH only allows for the execution of a single command. This time, we want to execute multiple commands on the remote machine. To complicate things even further, this time imagine that we have several different custom installations of Linux with the KILLALL and PS commands in different directories. We will add some code to find "ps", kill our process, and verify that our process was killed. The commands we want to execute on the remote system look like this:

```
#!/bin/bash
PATH_TO_PS=`which ps`; PATH_TO_KILLALL=`which killall`
# find our ps and killall commands and store their location
if [ $PATH_TO_PS == '' ] || [ $PATH_TO_KILLALL == '' ]; then
   # if we didn't find one of them, record an error
   echo "One of our commands is missing on $HOSTNAME"
else
   # can't find programs, use killall to kill the nc process
   $PATH_TO_KILLALL nc
   PROCCOUNT=`$PATH_TO_PS a | grep nc | wc -l`
   #we count the number of nc processes that are running
   if [ $PROCCOUNT -gt 1 ]
   #If we have more than 1 (grep will be one) then echo a
   warning
   then
      echo "Still running on $HOSTNAME"
   fi
fi
```



Our last topic in discussing Bash is interacting with a remote computer. Sometimes it is necessary to interact with the remote computer, rather than simply sending a single command. For example, if we are trying to login to a remote computer, you need your script to be able to "spawn" ssh, then have it wait for a login prompt before it transmits the username. After it enters the username, it needs to wait until the password prompt appears before it transmits the passwords. This problem can be solved using "expect".

For additional reading on this subject read this page: https://redsiege.com/ca/expect



Enable the OpenSSH server on your system with the following commands then answer the questions below.

You will then connect to your own system via SSH to simulate running the commands on a remote server.

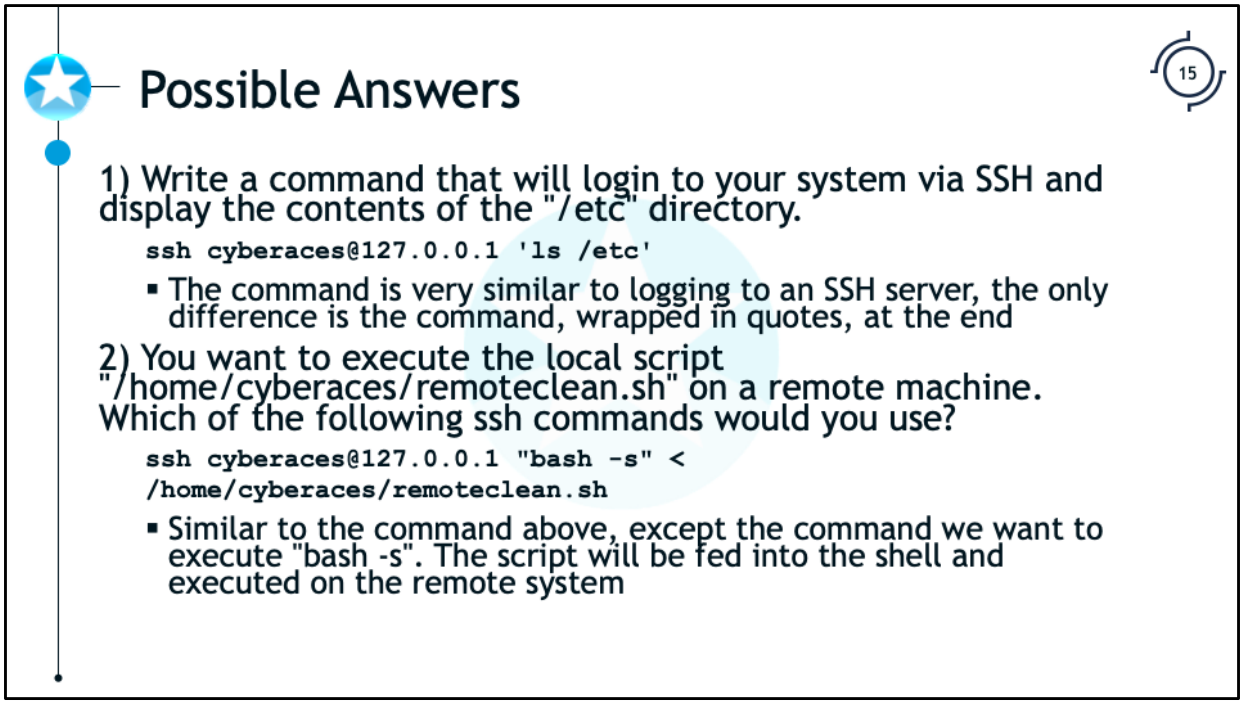
```
$ sudo systemctl start sshd
$ passwd
<enter a new password for yourself>
```

1) Write a command that will login to your system via SSH and display the contents of the "/etc" directory.

2) Create a script named "remoteclean.sh" in /home/cyberaces/ containing the command:

```
echo Clean Complete
```

Then write a command that will execute the local script "/home/cyberaces/remoteclean.sh" using SSH.



1) Write a command that will login to your system via SSH and display the contents of the "/etc" directory.

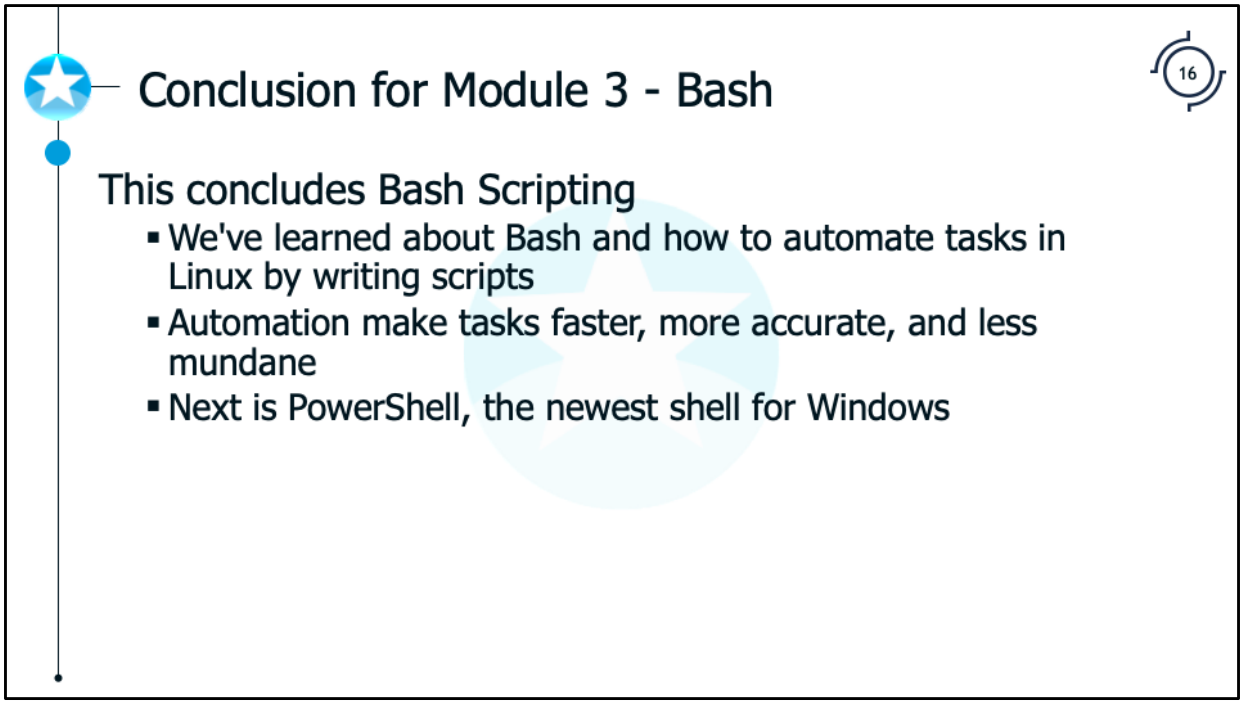
```
ssh cyberaces@127.0.0.1 'ls /etc'
```

The command is very similar to logging to an SSH server, the only difference is the command, wrapped in quotes, at the end.

2) You want to execute the local script "/home/cyberaces/remoteclean.sh" on a remote machine. Which of the following ssh commands would you use?

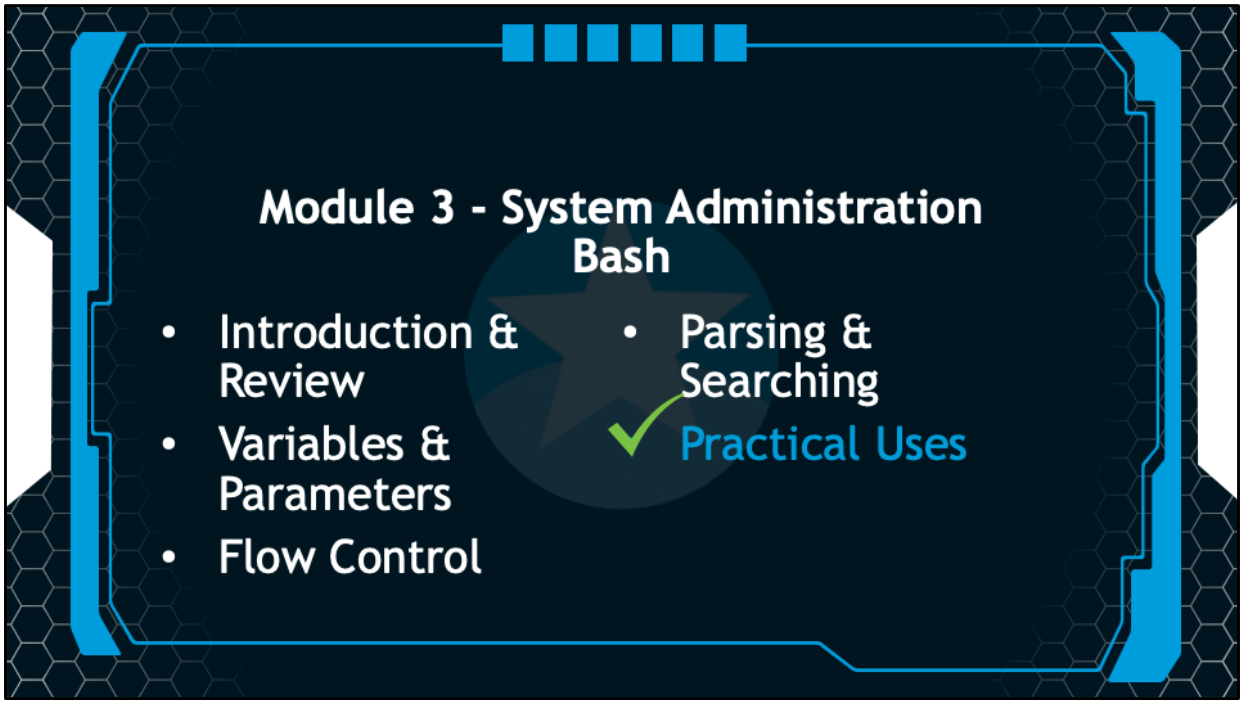
```
ssh cyberaces@127.0.0.1 "bash -s" < /home/cyberaces/remoteclean.sh
```

Similar to the command above, except the command we want to execute "bash -s". The script will be fed into the shell and executed on the remote system.



This concludes BASH Scripting. We've learned about BASH and how to automate tasks in Linux by writing scripts. Automation make tasks faster, more accurate, and less mundane.

The next module is on PowerShell.



This is the conclusion of the Bash Scripting Module.