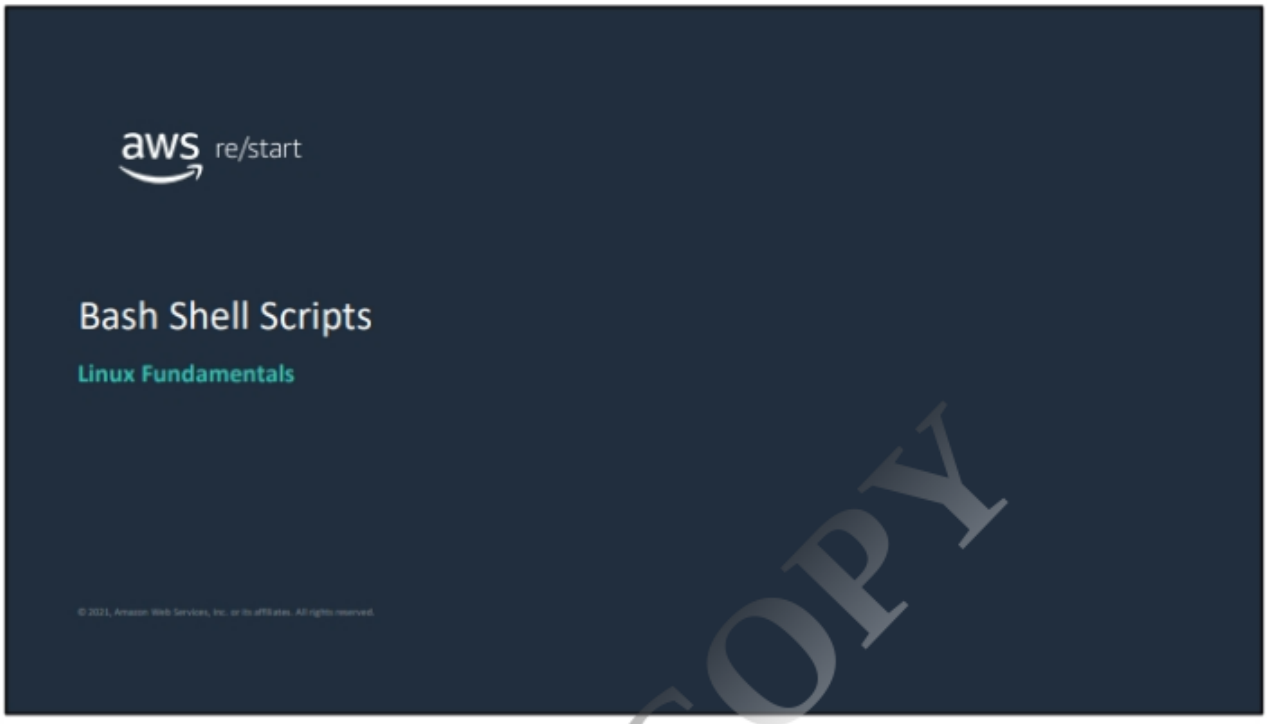
Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



Welcome to Bash Shell Scripts.

DO NOT COPY
bufetekaye.22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

What you will learn

At the core of the lesson

You will learn how to:

- Describe common tasks that are accomplished through shell scripts
- Describe basic commands that are frequently included in shell scripts
- Describe basic logical control statements that are frequently included in shell scripts
- Run a shell script

2 © 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

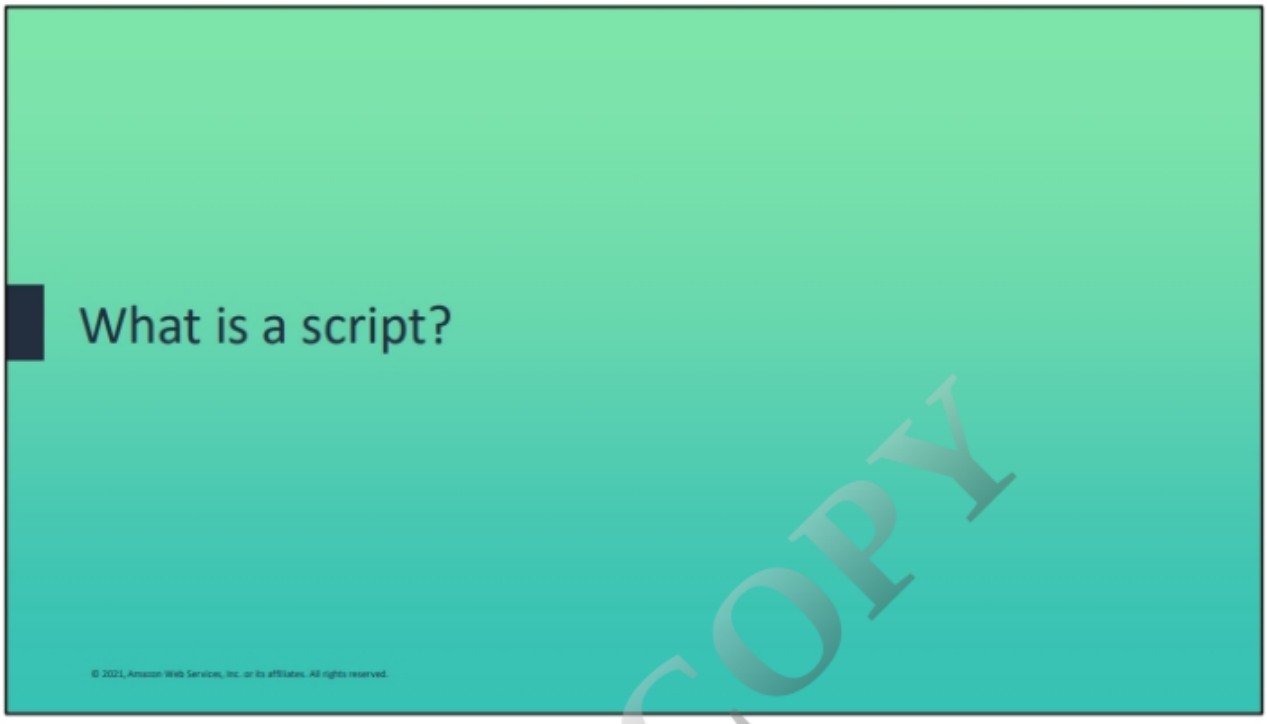
aws re/start

In this module, you will learn how to:

- Describe common tasks that are accomplished through shell scripts
- Describe basic commands that are frequently included in shell scripts
- Describe basic logical control statements that are frequently included in shell scripts
- Run a shell script

DO NOT COPY
bufetekaye.22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



This section defines a script and covers common tasks that you can accomplish by using shell scripts.

DO NOT COPY
bufetekaye.22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

What are scripts?

- Scripts are text files of commands and related data.
- When the text file is processed, the commands are run.
- Scripts can be set as scheduled tasks by using cron.
- Automation allows scripts to run more quickly than if they are run manually.
- Scripts are consistent due to automation removing the potential for manual errors.

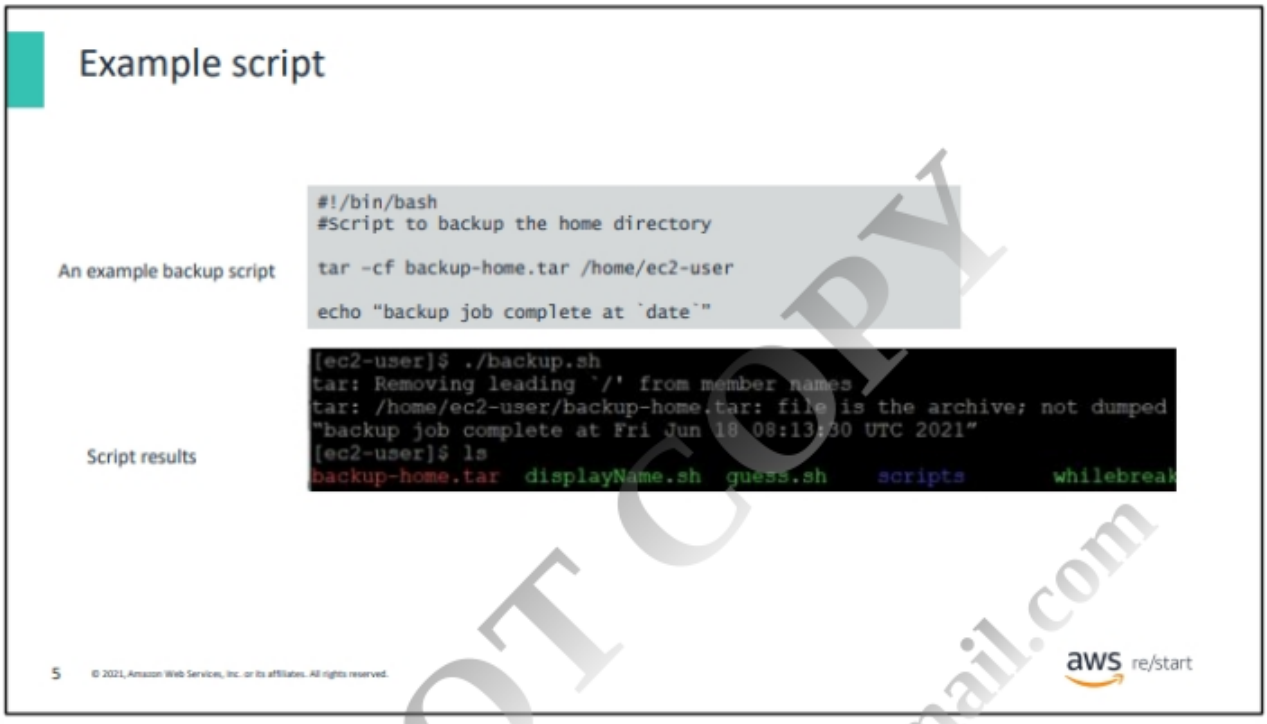
© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.
aws re/start

Common script tasks:

- Creating backup jobs
- Archiving log files
- Configuring systems and services
- Simplifying repetitive task
- Automating tasks

DO NOT COPY
bufetekaye.22@gmail.com

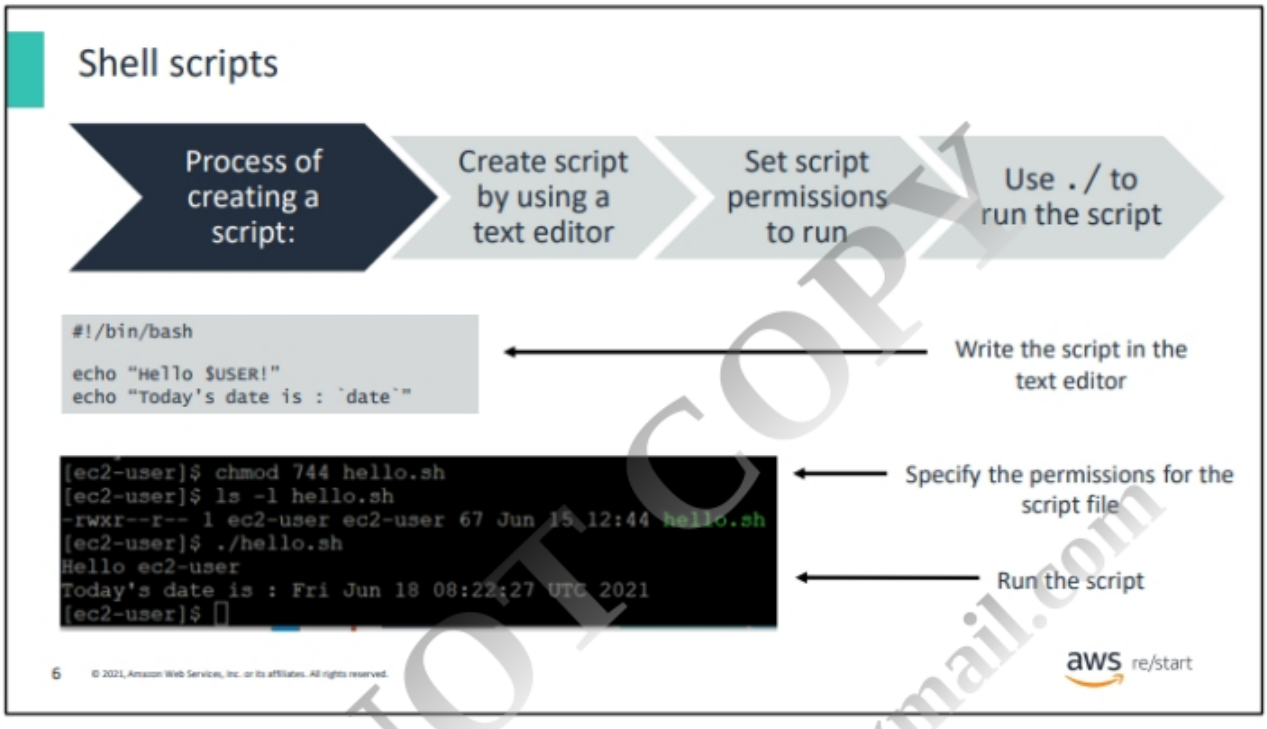
Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



This script creates a tar archive with the content of the folder /home/ec2-user.

DO NOT COPY
bufetekaye.22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

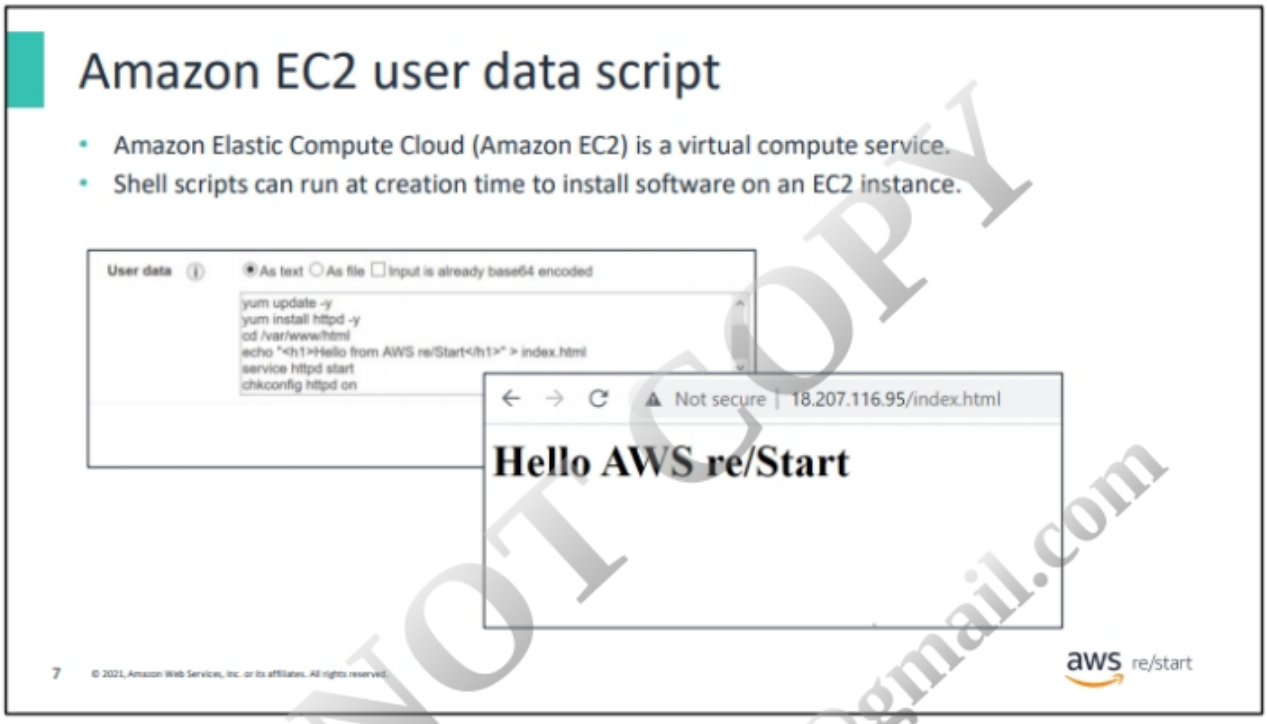


The process of creating a script follows these steps:

1. Create the script using a text editor.
2. Set the script permissions to run.
3. Use ./ to run the script.

DO NOT COPY
bufetekaye.22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



This screen capture is an example of a user shell script that runs when the EC2 instance is created.

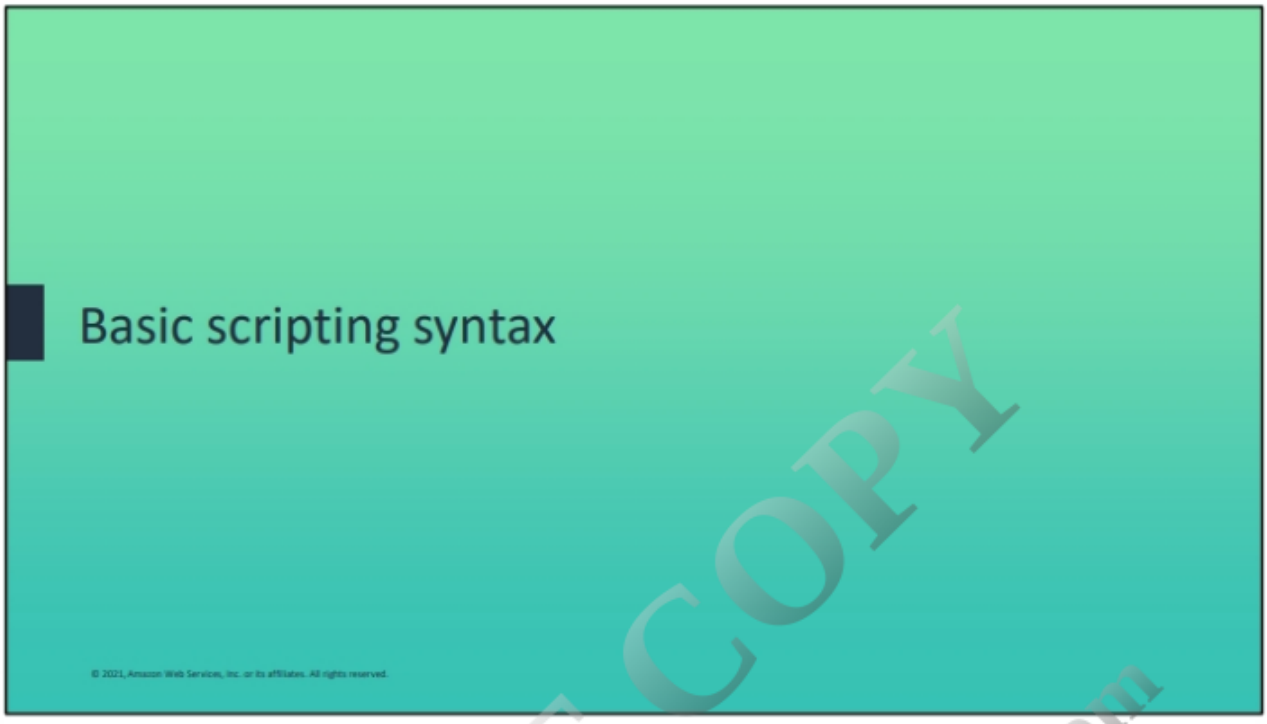
An HTTP web server is installed and run.

A basic HTML page is created.

The web server can be accessed via the public IP address of the EC2 instance.

DO NOT COPY
bufetekaye.22@gmail.com

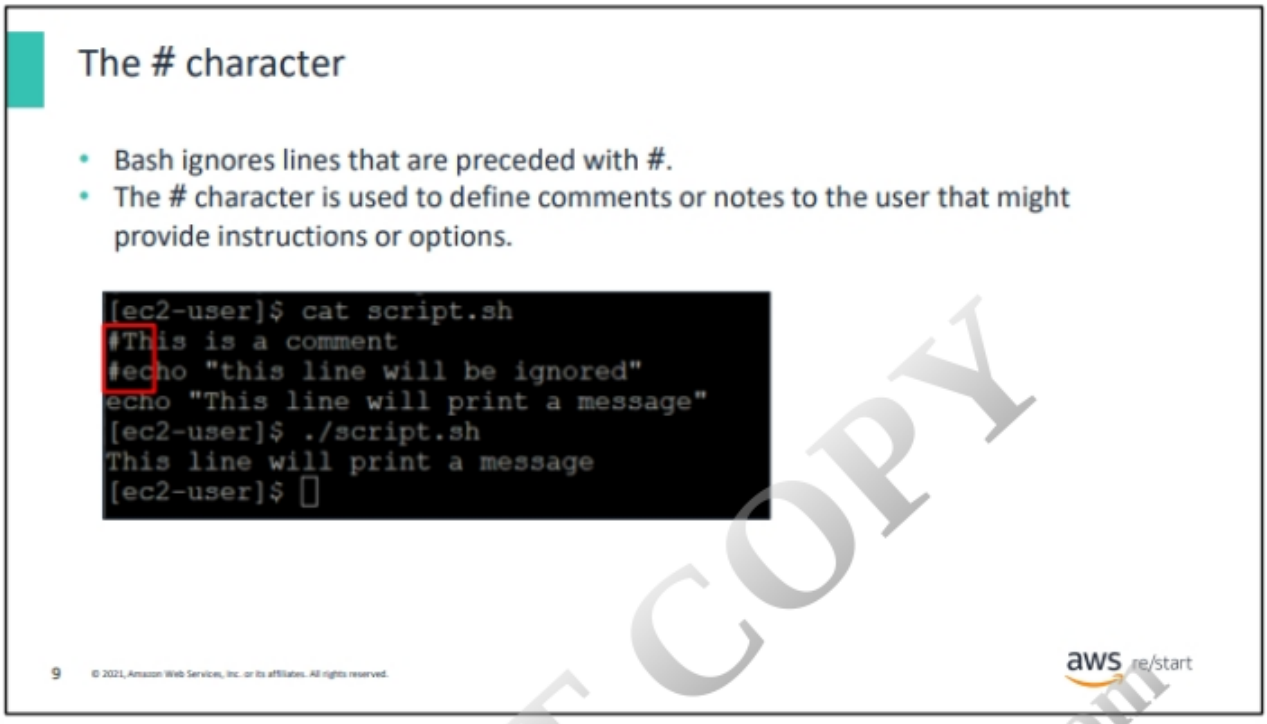
Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



This section introduces basic commands that are frequently included in shell scripts.

DO NOT COPY
bufetekaye.22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



Only the second echo command runs and displays a message.

DO NOT COPY
bufetekaye.22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

#!/bin/bash and #comments

- #! is referred to as a *shebang*.
- The first line defines the interpreter to use (gives the path and name of the interpreter).
- Scripts must begin with the directive for which shell will run them.
- The location and shell can be different.
- Each shell has its own syntax, which tells the system what syntax to expect.
  - Example: `#!/bin/bash`
- Use # to define comments, including the purpose of the script, author information, special directives for the script, examples, and others.

```
[ec2-user]$ cat hello.sh
#!/bin/bash

echo "Hello `whoami`"
echo "Today's date is : `date`"
[ec2-user]$
```

10 © 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws re/start

If the first line does not define the interpreter, the operating system will find one, usually the one defined for the current shell.

DO NOT COPY
bufetekaye.22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

Script documentation

- Some administrators create a script template, which contains all the relevant information and sections.
- The template might include the following:
  - Title
  - Purpose
  - Author's name and contact information
  - Special instructions or examples

```
#/bin/bash

#.....................
#Author : Jane Doe
# Date : 06/15/2021

#Description :Here is how you can document a script
#
#
#Usage :
# ./myScript.sh param1 [param2]
#param 1:
#param2:
#.....................
#Version: 2.0.1

#Declared variables

#.....................
#Scrit body
```

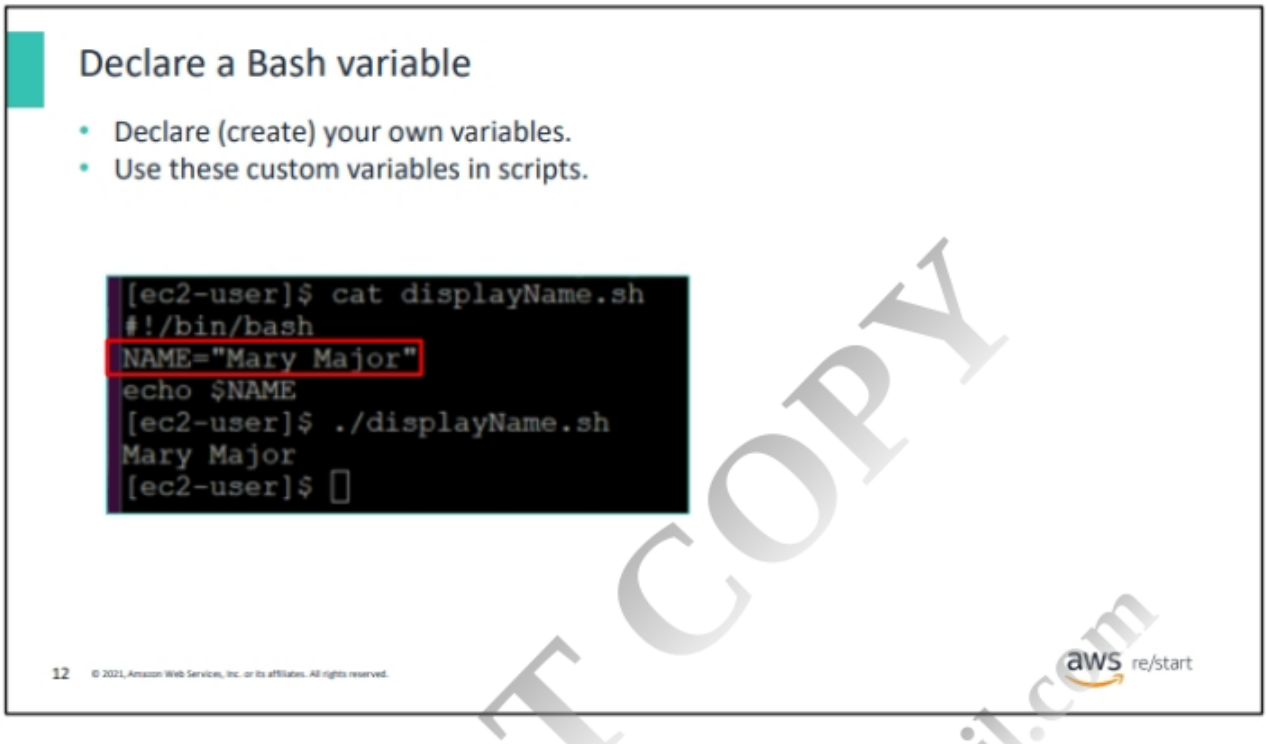
11 © 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws re/start

There is no exact format definition that scripts should follow.

DO NOT COPY
bufetekaye.22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



In the example, a variable NAME has been created.

DO NOT COPY
bufetekaye.22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

Useful commands

Command	Description
echo	Displays information on the console
read	Reads a user input
subStr	Gets the substring of a string
+	Adds two numbers or combine strings
file	Opens a file
mkdir	Creates a directory
cp	Copies files
mv	Moves or renames files
chmod	Sets permissions on files
rm	Deletes files, folders, etc.
ls	Lists directories

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.
aws re/start

This table lists the most useful commands that you can use in a script.

You can use all the shell commands that you saw earlier in the course, such as **grep, touch,** and **redirectors (>, >>, <, <<)**.

You can also use the shell script to create programs using statements such as if, else if, for, while, case, and create functions using the **function** keyword.

DO NOT COPY
bufetekaye.22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

Demonstration: Declaring a Variable in a Script

Overview

- Some variables are configured by default, but you can also declare your own variables. This functionality is often used when you write and run scripts. In this activity, you write a simple script to declare variables and use those variables in a practical use case.
- In this demonstration, a simple Bash script is created to demonstrate storing a user's name.

14 © 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws re/start

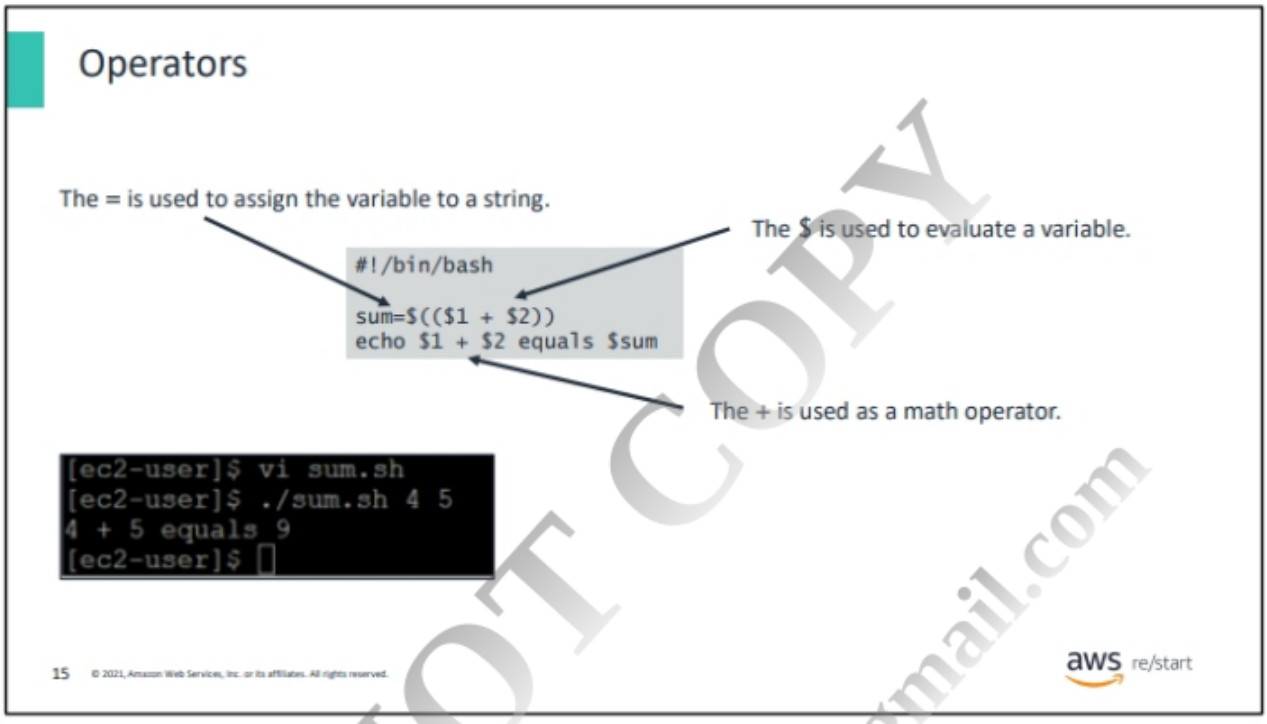
1. For this demonstration, create a file and enter the following text:
```
#!/bin/bash
echo "What is your name?“
read name
echo "Hello $name"
```

2. As soon as you finish, save the file and run the following command (for this example, the file is named *getname.sh*):
```
chmod +x getname.sh
```

3. Run the script by entering the following (for this example, the file is named *getname.sh*):
```
./getname.sh
```

DO NOT COPY
bufetekaye.22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

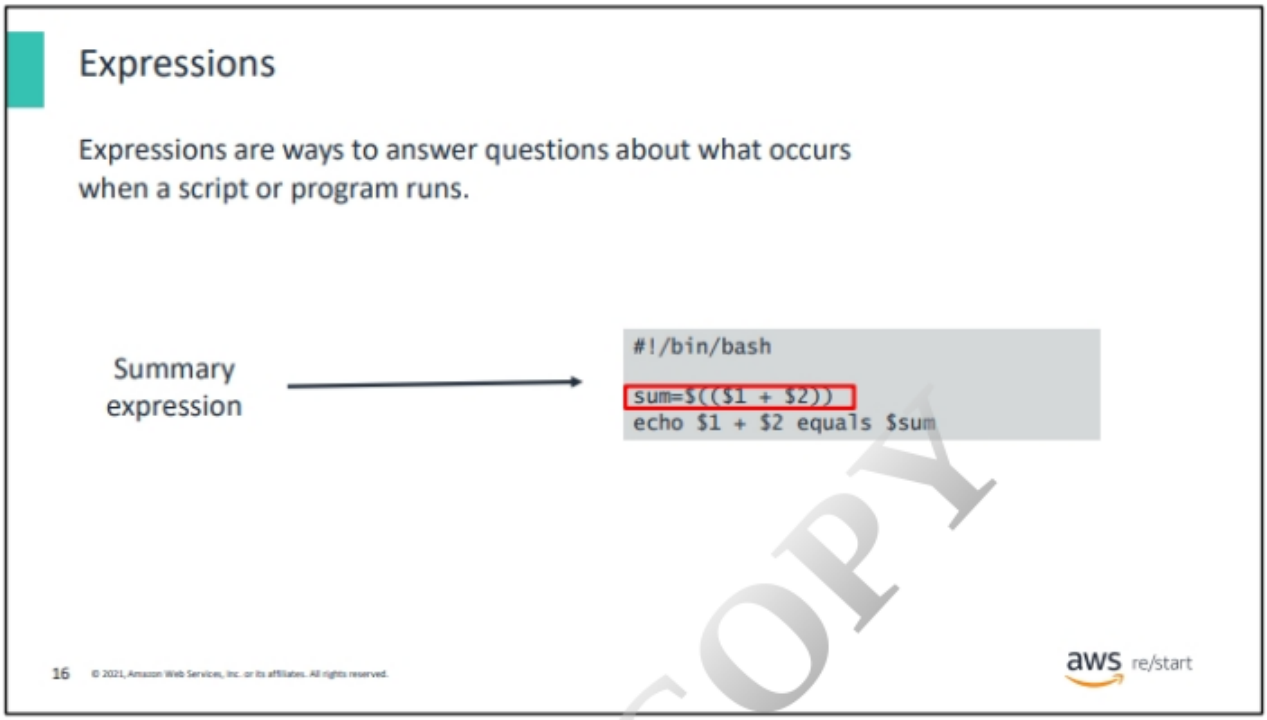


$1 and $2 are the parameters passed to the scripts.

You can also use the + operator to concatenate string.

DO NOT COPY
bufetekaye.22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



Expressions are ways to answer questions about what occurs when a script or program runs. Such questions might include the following:

- Who is running the script?
- What time is it?

The answer is usually assigned to a variable for reference later in the script. This example asks, "What is the sum of the two numbers that the user entered when they ran the script?"

The value is then stored in the **sum** variable.

DO NOT COPY
bufetekaye.22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

Conditional statements

Conditional statements allow for different courses of action for a script depending on the success or failure of a test.

In the following script, the `rm` command is run only if the `cp` command successfully completes.

```
#!/bin/bash
#Copy file1 to /tmp
#Delete file1 if the copy was
successful

if [ cp file1 /tmp ];
then
     rm file1
fi
```

```
[ec2-user]$ ls
delete.sh  displayName.sh  error.log   example.txt
Desktop    Documents       example.sh  file1
[ec2-user]$ ./delete.sh
[ec2-user]$ ls
delete.sh  displayName.sh  error.log   example.txt
Desktop    Documents       example.sh  hello.sh
[ec2-user]$ ls /tmp
file1
[ec2-user]$
```

`file1` does not exist anymore.

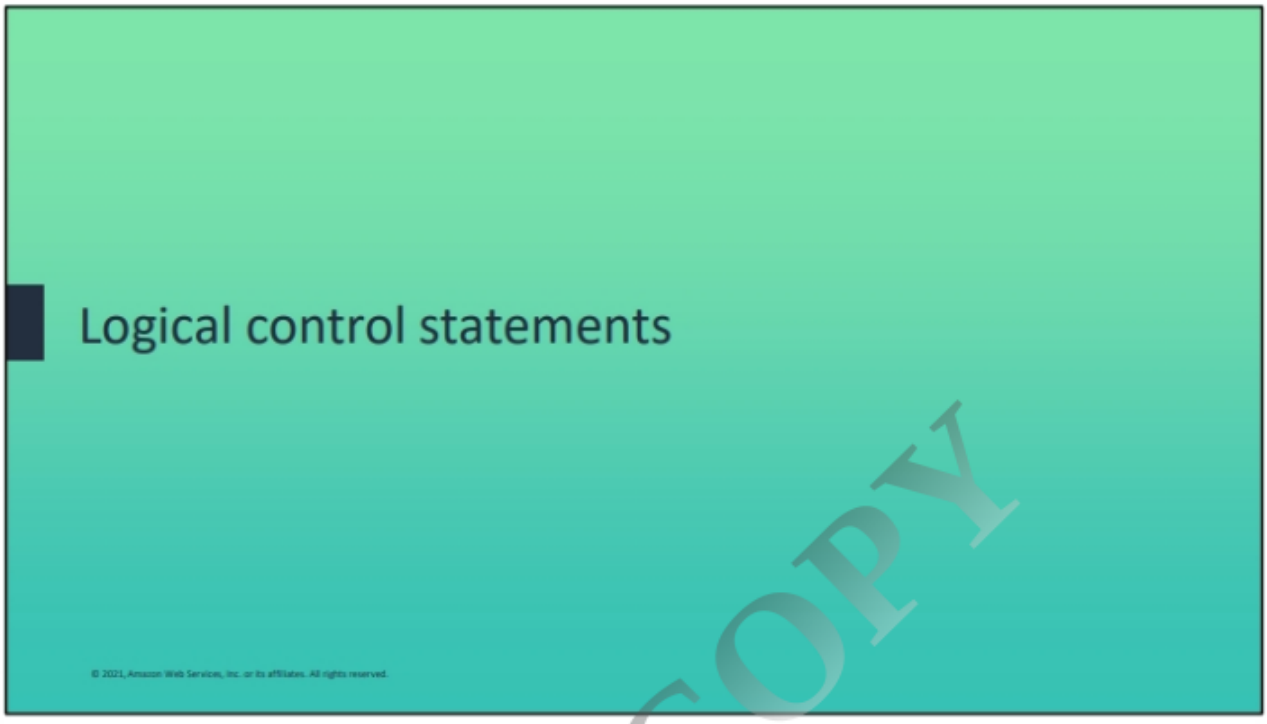
17 © 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws re/start

Conditional actions are dependent on outcomes.

DO NOT COPY
bufetekaye.22@gmail.com

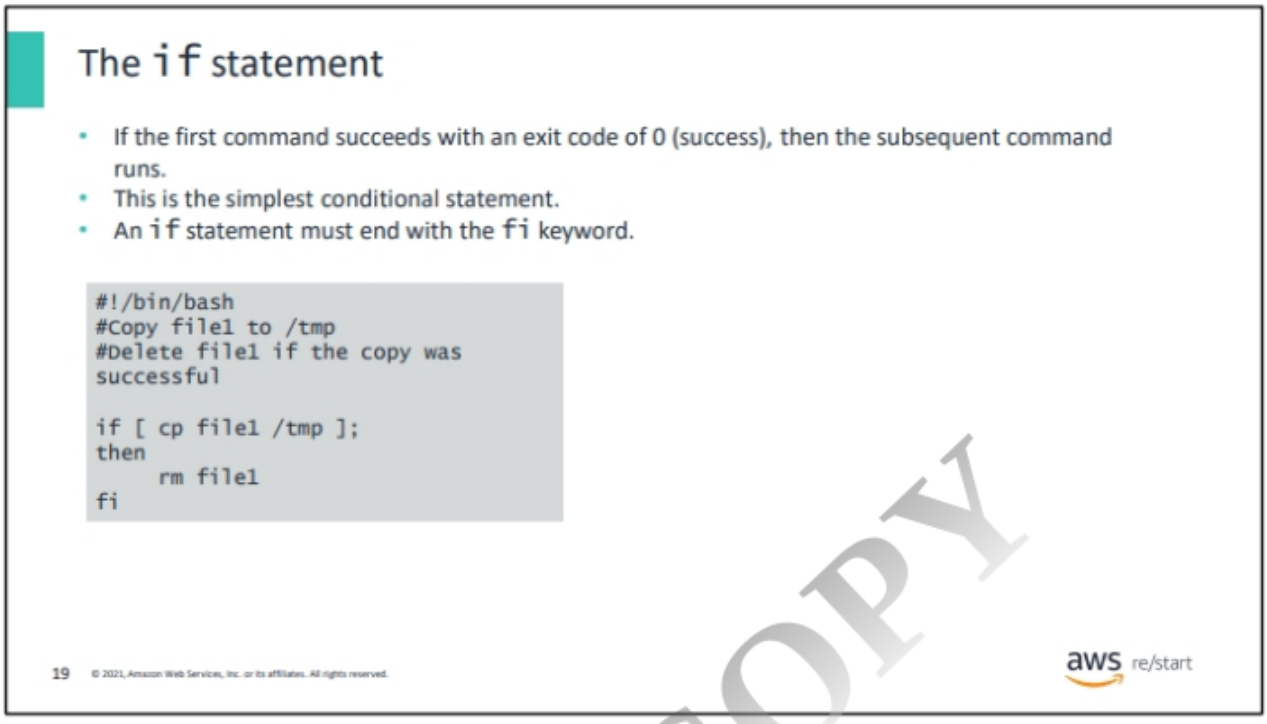
Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



The section focuses on basic logical control statements that are frequently included in shell scripts, such as if, if-else, if-elif-else, and test.

DO NOT COPY
bufetekaye.22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



An if statement is written as follows:

if <condition>
then
<command>
fi

Or inline: if <condition>; then <command> fi

The semicolon (;) is not mandatory if not writing inline.

Indentation is used for better readability but is not required.

DO NOT COPY
bufetekaye22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

The `if - else` statement

- Defines two courses of action:
  - If the condition is true (`then`)
  - If the condition is false (`else`)

`else` statement

```
#!/bin/bash
#Copy file1 to /tmp
#Delete file1 if the copy was
successful

if [ cp file1 /tmp ];
then
     rm file1
else
     echo "No such file."
fi
```

The `else` statement is invoked because `file1` does not exist.

```
[ec2-user]$ ./delete.sh
cp: cannot stat 'file1': No such file or directory
No such file.
[ec2-user]$
```

20 © 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws re/start

An if-else statement is written as follows:

if <condition>
then
<command>
else
<other command>
fi

DO NOT COPY
bufetekaye.22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

The `if - elif - else` statement

- Defines three courses of action:
  - If the condition is true (`then`)
  - Else if the condition is false but another condition is true (`then`)
  - If the second condition is false (`else`)

```
#!/bin/bash
#Compares $1 and $2

if [$1 -gt $2 ];
then
     echo "the first number is greater then the second number"
elif [ $1 -lt $2 ];
then
     echo "the second number is greater then the first number"
else
     echo "the two numbers are equal"
fi
```

21 © 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws re/start

DO NOT COPY
bufetekaye.22@gmail.com

The example compares two numbers passed as parameters to the script:

- **-gt** means greater than
- **-lt** means lower than

An if-elif-else statement is written as follows:

if <condition>
then
<command>
elif <other condition>
then
<other command>
else
<default command>
fi

- You can embed if-elif-else statements.
- You can access all local objects of its immediately enclosing function and also of any function or functions that enclose that function.
- Nesting is theoretically possible to unlimited depth.

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

The `test` command

- Checks file types and compare values
- Conditions are set, and then the test *exits* with a 0 for true and a 1 for false.
- Syntax: `test <EXPRESSION>`

Test whether this comparison is true or false.

```
#!/bin/bash
#Compares two values

test  100 -lt 99 && echo "Yes "  || echo "No
```

The comparison is false.

```
[ec2-user]$ ./compare.sh
No
[ec2-user]$
```


© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws re/start

You use the test command to compare values.

DO NOT COPY
bufetekaye.22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

Integer comparison operators

Comparison operators compare two variables or quantities.

- `-eq` is equal to: `if [ "$a" -eq "$b" ]`
- `-ne` is not equal to: `if [ "$a" -ne "$b" ]`
- `-gt` is greater than: `if [ "$a" -gt "$b" ]`
- `-ge` is greater than or equal to: `if [ "$a" -ge "$b" ]`
- `-lt` is less than: `if [ "$a" -lt "$b" ]`
- `-le` is less than or equal to: `if [ "$a" -le "$b" ]`
- < is less than (within double parentheses): `(("$a" < "$b"))`
- <= is less than or equal to (within double parentheses): `(("$a" <= "$b"))`
- > is greater than (within double parentheses): `(("$a" > "$b"))`
- >= is greater than or equal to (within double parentheses): `(("$a" >= "$b"))`

23 © 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

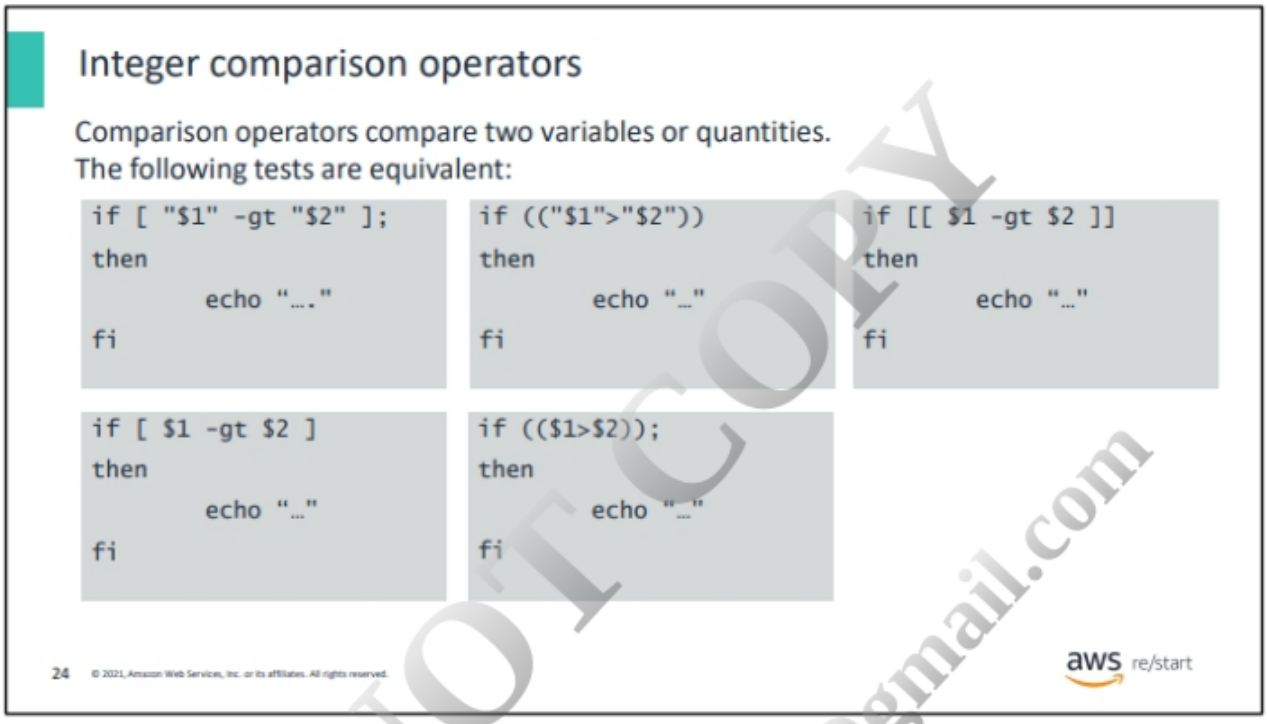
aws re/start

In these examples, you can see that brackets or double parentheses surround the condition.

- ((are for numerical comparison:
  - **(($a < $b))** is equivalent to **[$a –lt $b]**
- You will also see some conditions using double brackets:
  - **if [[$a –lt $b]]**
  - This is basically a notation equivalent to a simple bracket with enhanced features, but this goes beyond this course.
- Note also that " are not mandatory: **if ["$a" -lt "$b"]** is equivalent to **if [$a -lt $b]**

DO NOT COPY
bufetekaye.22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



- " are not mandatory: **if ["$a" -lt "$b"]** is equivalent to **if [$a -lt $b]**
- The semicolon (;) is mandatory only if the test is written inline: **if [$a -lt $b]; then**

...

DO NOT COPY
bufetekaye.22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

String comparison operations

- = or == is equal to
  - if ["$a" = "$b"]
  - if ["$a" == "$b"]
- != is not equal to
  - `if [ "$a" != "$b" ]`
  - This operator uses pattern matching within a `[[ ... ]]` construct.
- < is less than, in ASCII alphabetical order
  - `if [[ "$a" < "$b" ]]`
  - `if [ "$a" \< "$b" ]`
  - Note that the < must be escaped in a [ ] construct
- > is greater than, in ASCII alphabetical order
  - `if [[ "$a" > "$b" ]]`
  - `if [ "$a" \> "$b" ]`
  - Note that the > must be escaped in a [ ] construct.
- `-z` string is null (that is, it has zero length)
- `-n` string is not *null*

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws re/start

Familiarize yourself with these string comparison operations.

DO NOT COPY
bufetekaye.22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

String comparison operations example

```
#!/bin/bash
#Compares letters $1 and $2

if ["$1" == "$2" ];
then
     echo "letters are the same"
elif [ $1 \< $2 ];
then
     echo "the first letter is before the second letter"
else
     echo "the second letter is before the first letter"
```

```
[ec2-user]$ ./letters.sh a g
the first letters is before the second letter
[ec2-user]$ ./letters.sh a a
letters are the same
[ec2-user]$
```

26 © 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

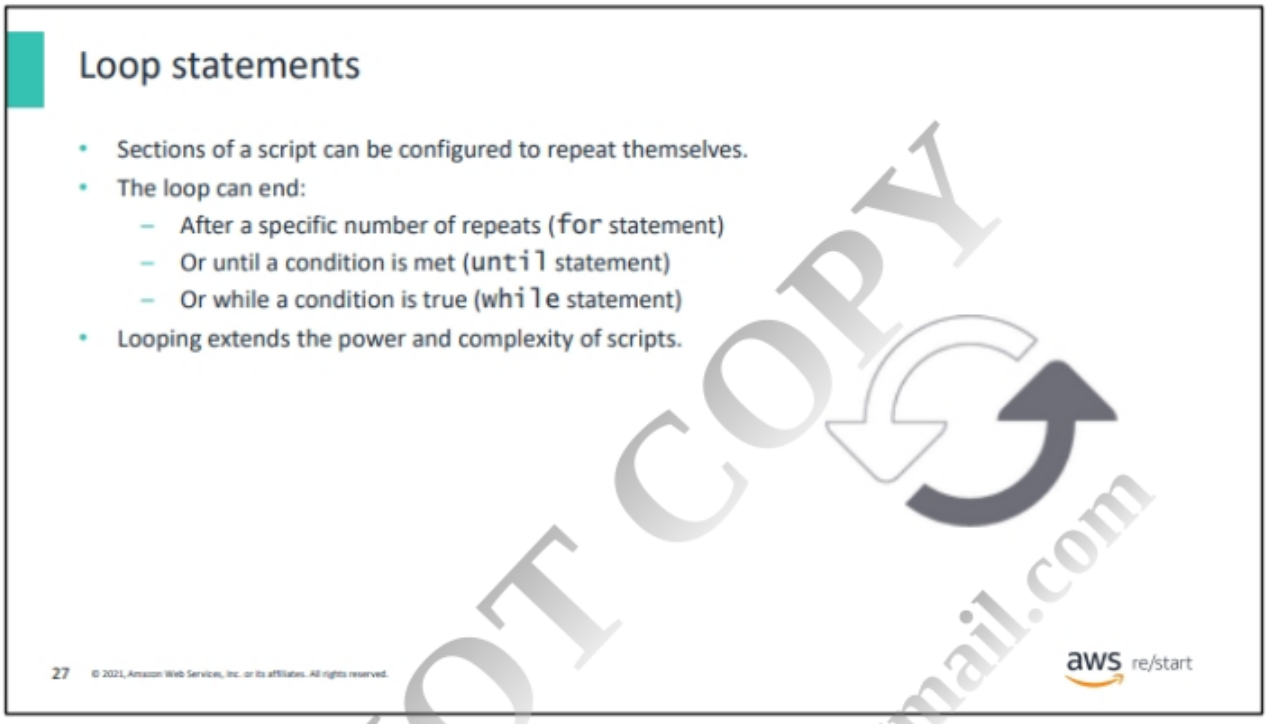
aws re/start

In the previous slide, $a and $b were used for generic comparisons.

Here you use $1 and $2 to match the first and second parameters passed to the script at runtime: **a and g**.

DO NOT COPY
bufetekaye.22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



Loops provide you the ability to repeat sections of a script.

Loops can end:

- After a specific number of repeats (**for** statement)
- Until a condition is met (**until** statement)
- While a condition is true (**while** statement)

DO NOT COPY
bufetekaye22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

The for statement

- Loops the command a specified number of times
- Bracketed by do and done

```
#!/bin/bash
#The for loop

for x in 1 2 3 4 5 a b c d
do
     echo "the value is $x"
done
```

```
[ec2-user]$ ./forloop.sh
the value is 1
the value is 2
the value is 3
the value is 4
the value is 5
the value is a
the value is b
the value is c
the value is d
[ec2-user]$
```

28 © 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws re/start

To loop a command a specific number of times, use the for statement.

DO NOT COPY
bufetekaye.22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

The `until` statement

- Similar to the `while` statement
- Runs code until a condition becomes true
- Bracketed by `until` and `done`

```
#!/bin/bash
#The until loop

counter=1
until  [ $counter -gt 10 ];
do
     echo  $counter
     ((counter++))
done
echo "loop exited"
echo "counter equals $counter
```

```
[ec2-user]$ ./untilloop.sh
1
2
3
4
5
6
7
8
9
10
loop exited
counter equals 11
[ec2-user]$
```

29 © 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws re/start

The until loop works the same way as the while loop.

In the example, the loop loops until the counter reaches 10 and then exits.

DO NOT COPY
bufetekaye.22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

The `while` statement

- Continues running the script as long as the specified condition is true
- Bracketed by while and done

```
#!/bin/bash
#The while loop

counter=1
while  [ $counter -le 10 ];
do
     echo  $counter
     ((counter++))
     if [ counter == $1 ];
     then
          break
     fi
done
echo "loop exited"
echo "counter equals $counter"
```

```
[ec2-user]$ ./whileloop.sh
1
2
3
4
5
6
7
8
9
10
loop exited
counter equals 11
[ec2-user]$
```


© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws re/start

The while loop runs as long as the counter is lower or equal to 10:

- It displays the value of the counter.
- It increments the counter by 1 (counter++ means counter = counter+1).
- When the counter reaches 11, the loop exits.

DO NOT COPY
bufetekaye.22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

Loop control statements

- Used with loops to better manage their conditions:
  - `break`: Stop running the entire loop.
  - `continue`: If a condition is met, break out of loop. The loops keeps running
- Example: If a specified value is discovered, then stop looping and continue through the script.
- Used in nested loops to continue to the next part of loop.

```
[ec2-user]$ ./whilebreakloop.sh 5
1
2
3
4
loop exited
counter equals 5
[ec2-user]$
```

```
#!/bin/bash
#The break statement

counter=1
while  [ $counter -le 10 ];
do
     echo  $counter
     ((counter++))
     if [ $counter == $1 ];
     then
          break
     fi
done
echo "loop exited"
echo "counter equals $counter
```

31 © 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws re/start

In this example, the while loop exits before the condition of the loop is met if the counter reaches the value passed as a parameter.

- The parameter ($1) is 5.
- The **if** condition is met when counter == 5.
- The **break** keyword is reached.
- The loop exits.

DO NOT COPY
bufetekaye.22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

Loop control statements

```
#!/bin/bash
#The continue statement

counter=1
while  [ $counter -le 10 ];
do
     echo  $counter
     ((counter++))
     if [ $counter == $1 ];
     then
          echo "condition met"
          continue
          echo "after continue"
     fi
     echo "loop keeps going"
done
echo "loop exited"
echo "counter equals $counter"
```

```
[ec2-user]$ ./whilecontinueloop.sh 5
1
loop keeps going
2
loop keeps going
3
loop keeps going
4
if condition met
5
loop keeps going
6
loop keeps going
7
loop keeps going
8
loop keeps going
9
loop keeps going
10
loop keeps going
loop exited
counter equals 11
[ec2-user]$
```


© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

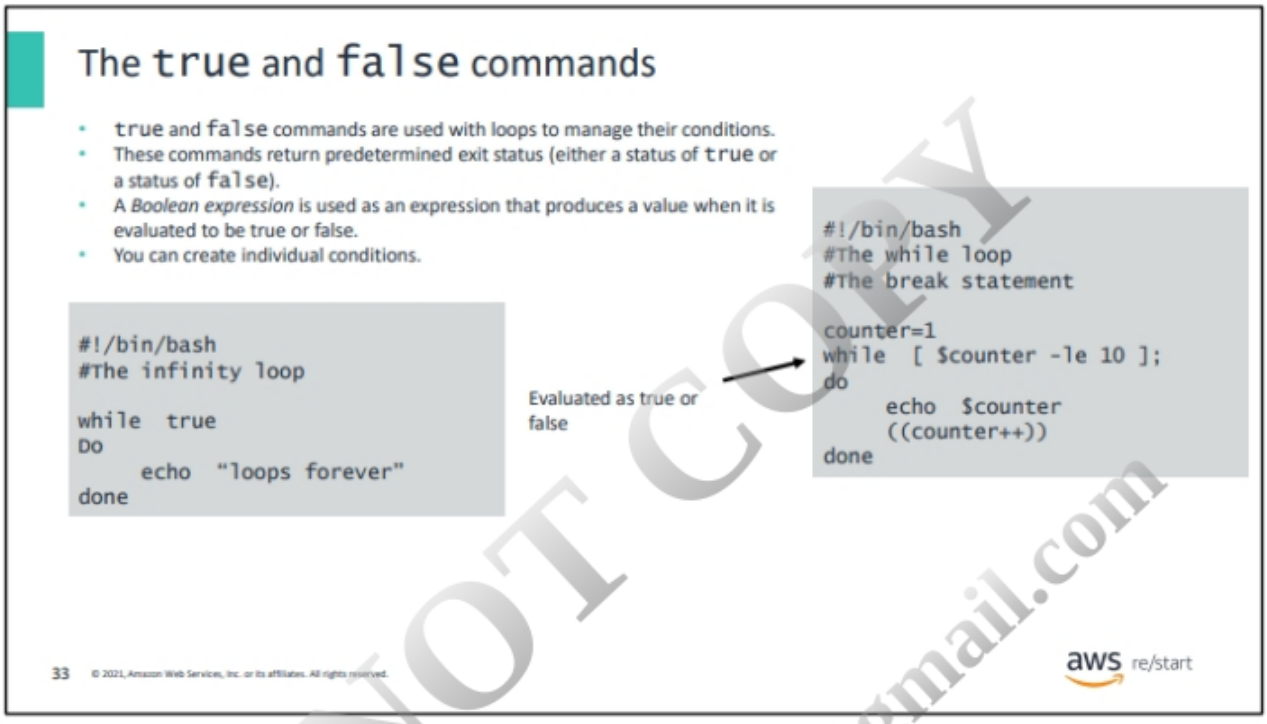
aws re/start

The **continue** keyword exits the if statement **(after continue** is not displayed), but the loop keeps going until the loop condition is met.

- The parameter is 5.
- The if condition is met when counter == 5.
- The **continue** keyword is reached.
- The if statement exits.
- The loop keeps running until **counter == 10** (original while condition).

DO NOT COPY
bufetekaye.22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



DO NOT COPY
bufetekaye.22@gmail.com

Return will be explained when working on functions.

An example of a **while true** is as follows:

```
#!/bin/bash

while true
Do
        echo "Enter a value"
        read val
        if [ $val –eq 5 ]
        break
        fi
        echo "wrong guess!"
done
Echo "good guess!"
```

It loops forever and asks the user to enter a value. If the user enters the right value (5), it exits the loop using a **break** statement.

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

The `exit` command

- Causes script to stop running and exit to the shell if the previous portion fails
- Useful in testing
- Can return code status. Each code can be associated to a specific error
- For example:
  - `exit 0`: the program has completed without any error
  - `exit 1`: the program has an error
  - `exit n`: the program has a specific error
- `$?` is a command to get the processing status of the last command that ran

```
#!/bin/bash

touch myfile.txt
if [ $? -eq 0];
then
     echo "file created"
     exit 0
else
do
     echo  "error when creating the file"
     exit 1
```

34 © 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

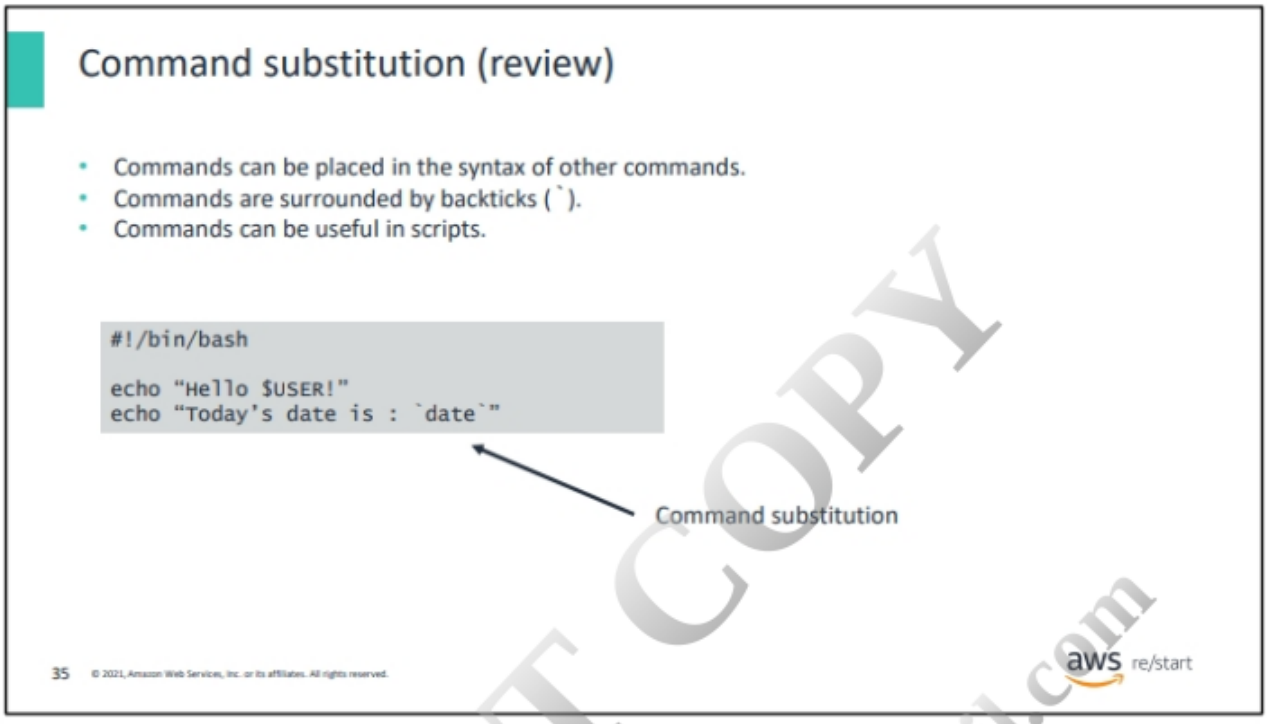
aws re/start

This script creates a file using the touch command.

It tests the code status of the touch command and exits accordingly.

DO NOT COPY
bufetekaye.22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



Command substitutions are useful in scripts.

DO NOT COPY
bufetekaye.22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

Arguments

- Arguments are values that you want the script to act on.
- Arguments are passed to the script by including them in a script invocation command separated by spaces.
- For example, $1 is the first argument, and $2 is the second argument.

```
#!/bin/bash

sum=$(($1+$2))
echo $sum
```

```
[ec2-user]$ ./math.sh 3 8
11
[ec2-user]$
```

36 © 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

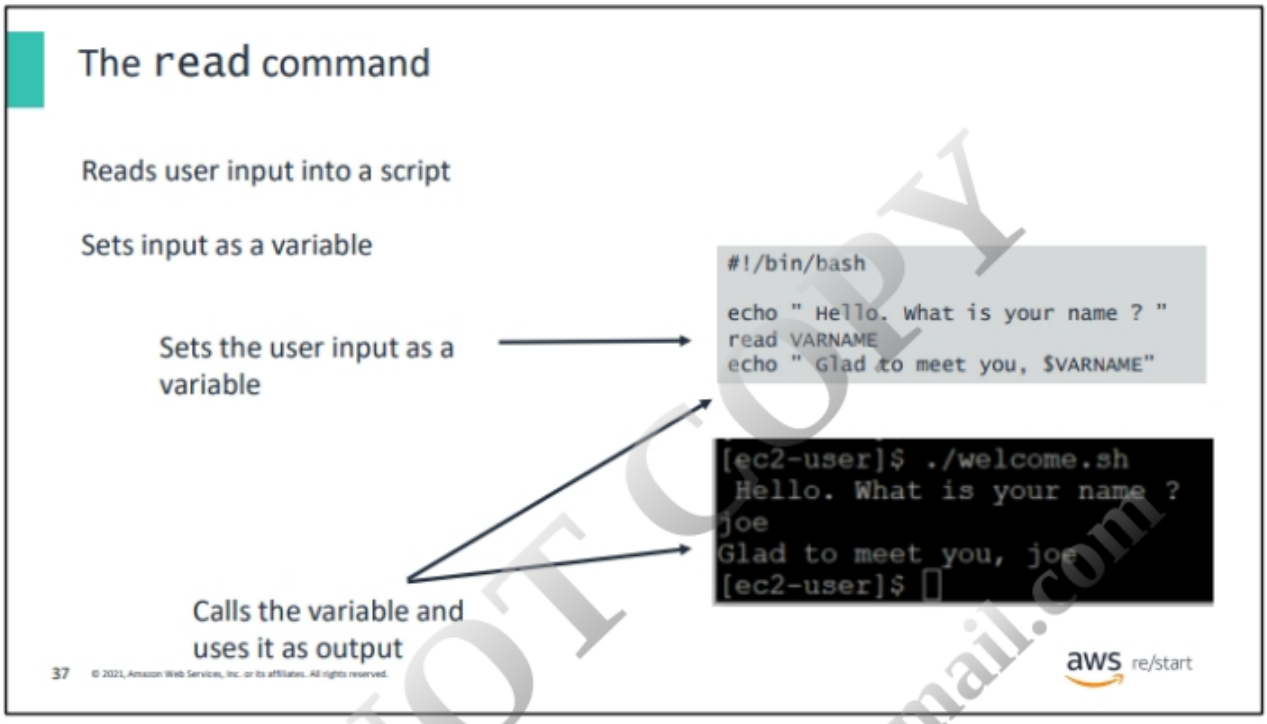
aws re/start

In this example, the arguments input by the user are 3 and 8.

Therefore, $1 is equal to 3 and $2 is equal to 8.

DO NOT COPY
bufetekaye.22@gmail.com

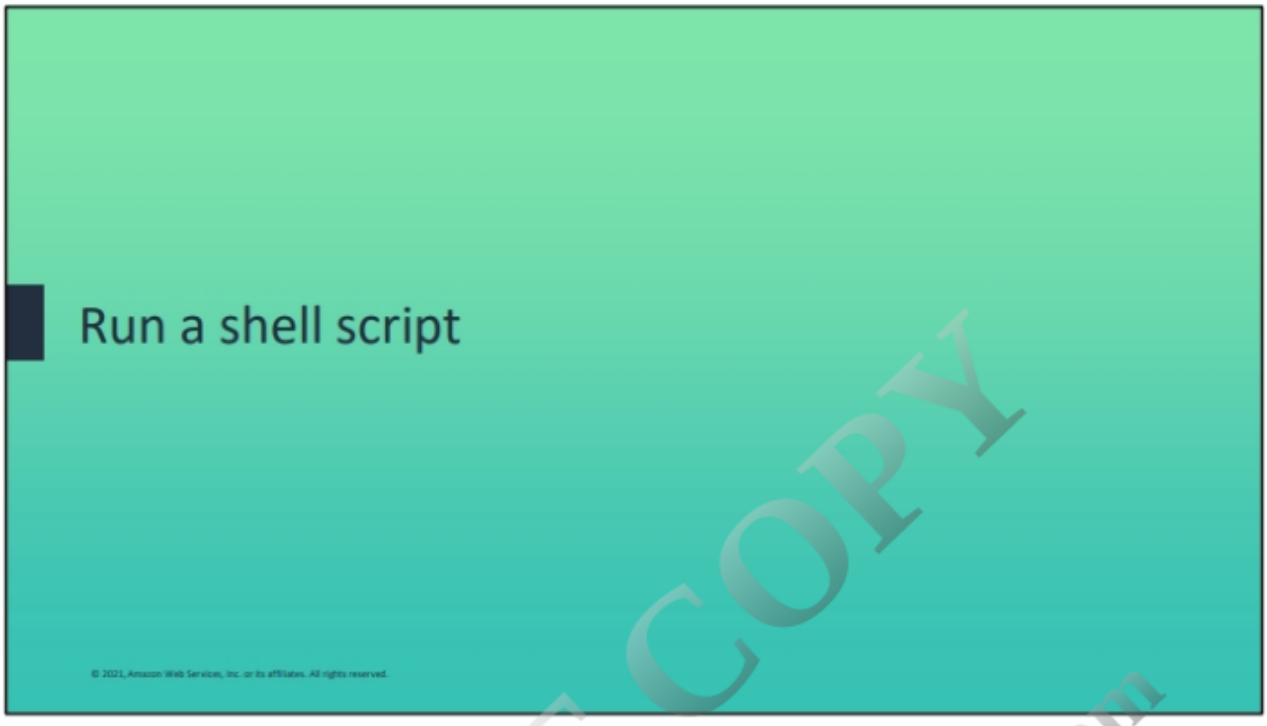
Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



Read incorporates the user's input as a variable into a script.

DO NOT COPY
bufetekaye.22@gmail.com

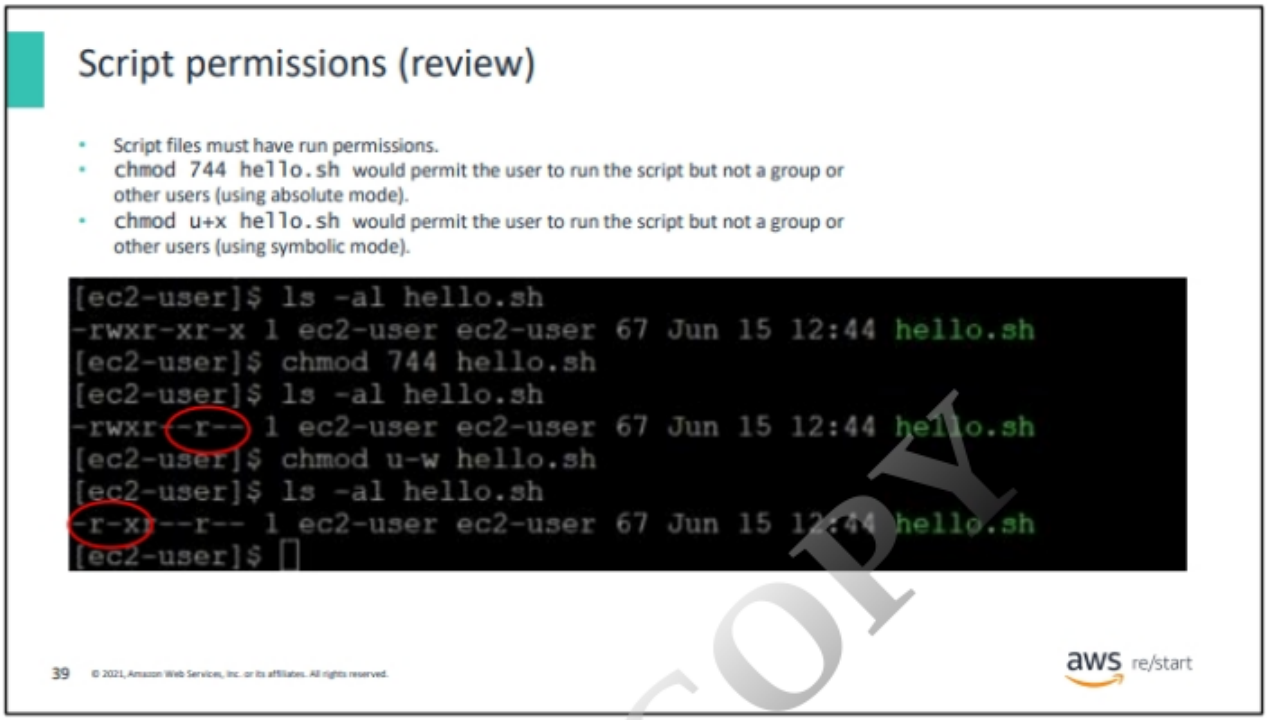
Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



In this section, you run a shell script.

DO NOT COPY
bufetekaye.22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



chmod 744 is:

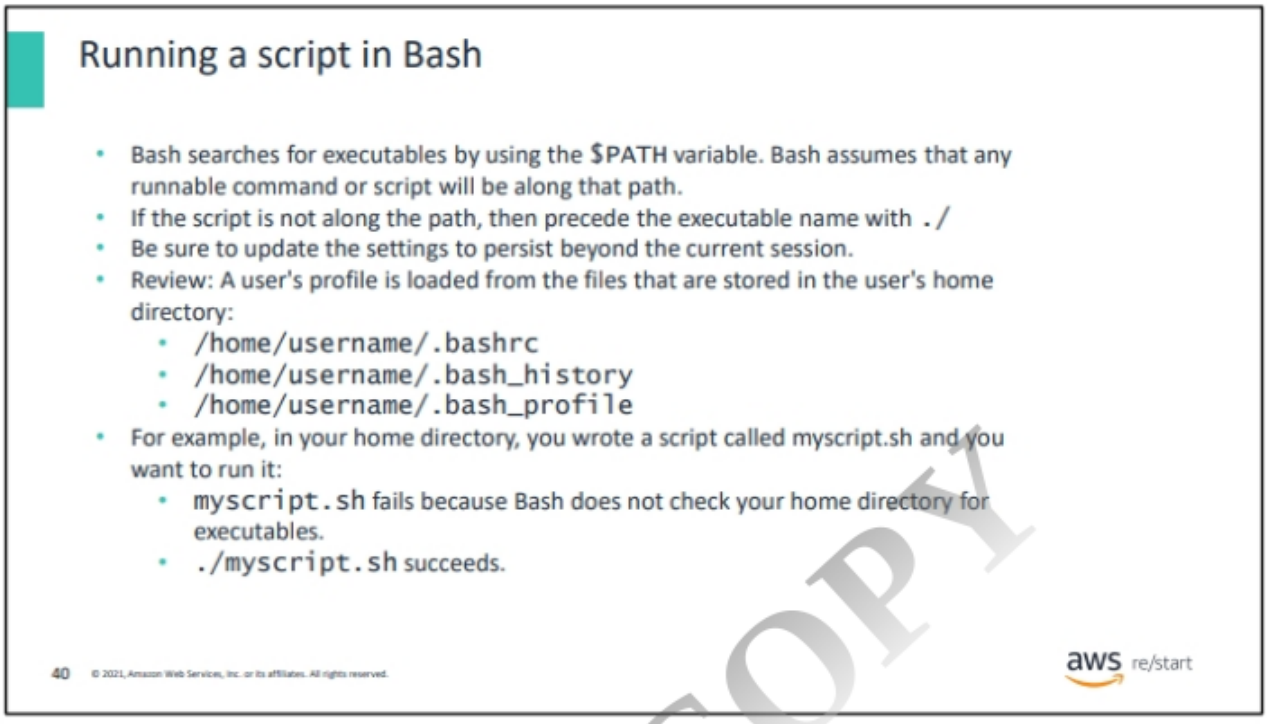
- Read (r) for the owner, the group of the owner, and others
- Write (w) for the owner only
- Execute (x) for the owner only
- Hence rwxr- -r- -

The second command, chmod u-w, removes the write right for the owner:

rwxr- -r- - becomes r-xr- -r- -

DO NOT COPY
bufetekaye.22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



Review this information so that when you run a script in Bash you don't have any issues.

DO NOT COPY
bufetekaye.22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

Run a script and ./ (review)

- Bash checks for executables along $PATH, which does not (and should not) include home directories.
- To run a script that is not stored on $PATH, use ./ before the script.
- Example: ./hello.sh
  - This example assumes that the script is in the current directory.
  - Otherwise, you must provide the absolute or relative path, or maybe an alias (for example, $HOME).
  - cron jobs always need the full path.

```
[ec2-user]$ hello.sh
-bash: hello.sh: command not found
[ec2-user]$ ./hello.sh
Hello ec2-user
Today's date is : Fri Jun 18 07:43:55 UTC 2021
[ec2-user]$ /home/ec2-user/hello.sh
Hello ec2-user
Today's date is : Fri Jun 18 07:44:12 UTC 2021
[ec2-user]$
```

41 © 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

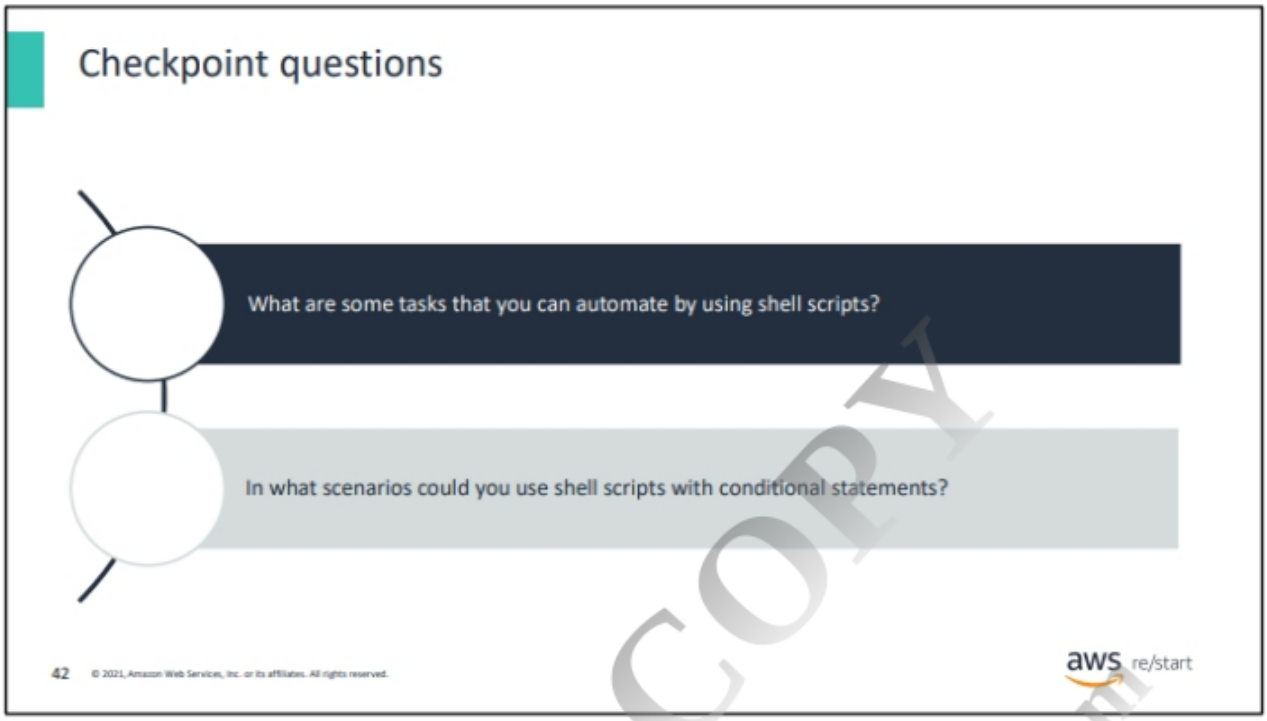
aws re/start

The home folder is not in **$PATH**

- **hello.sh** fails.
- **./hello.sh** succeeds because the current folder is the folder containing the script.
- Using the full path **/home/ec2-user/hello.sh** works from any folder.

DO NOT COPY
bufetekaye.22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

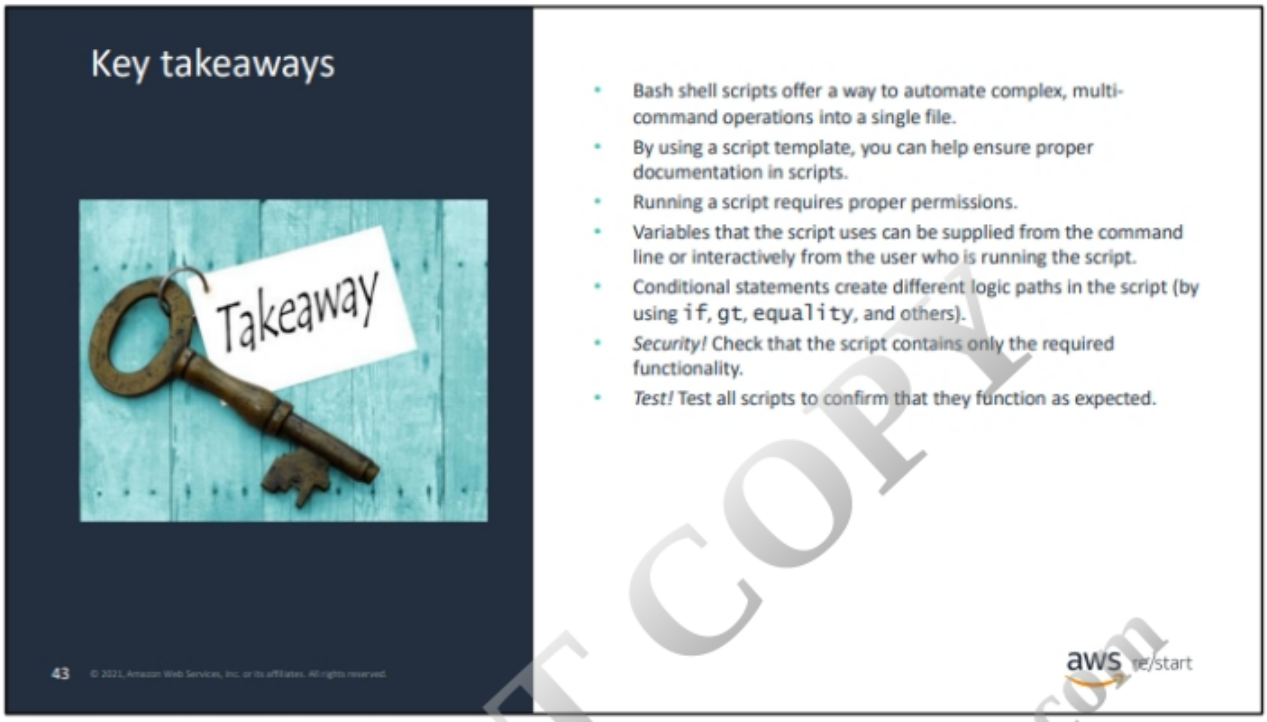


1. Some common tasks for shell scripts include:
   - Backing up important information
   - Moving files to storage locations so that only the newest files are visible
   - Finding duplicate files where thousands of files exist

2. Shell scripts with conditional statements can be used when:
   - The script asks the user a question that has only a few answer choices
   - Deciding whether the script must be run
   - Ensuring that a command ran correctly and taking action if it failed

DO NOT COPY
bufetekaye.22@gmail.com

Printed by: bufetekaye.22@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



Some key takeaways from this lesson include:

- Bash shell scripts offer a way to automate complex, multi-command operations into a single file.
- By using a script template, you can help ensure proper documentation in scripts.
- Running a script requires proper permissions.
- Variables that the script uses can be supplied from the command line or interactively from the user who is running the script.
- Conditional statements create for different logic paths in the script (by using **if**, **gt**, equality, and others).
- ***Security!*** Check that the script contains only the required functionality.
- ***Test!*** Test all scripts to confirm that they function as expected.

DO NOT COPY
bufetekaye.22@gmail.com