



Bash Shell 스크립트

Linux 기본 사항

학습 내용

강의 핵심 내용

학습 내용:

- 셀 스크립트로 수행되는 일반 태스크를 설명합니다.
- 셀 스크립트에 자주 포함되는 기본 명령을 설명합니다.
- 셀 스크립트에 자주 포함되는 기본 논리 제어 스테이트먼트를 설명합니다.
- 셀 스크립트를 실행합니다.





스크립트란?

스크립트란?

- 스크립트는 명령 및 관련 데이터의 텍스트 파일입니다.
- 텍스트 파일이 처리되면 명령이 실행됩니다.
- 스크립트는 크론(cron)을 사용하여 계획된 태스크로 설정됩니다.
- 스크립트는 수동으로 실행하기보다 자동화하여 실행하면 속도가 더 빨라집니다.
- 자동화하면 잠재적인 수동 오류를 제거하여 스크립트 일관성이 유지됩니다.

스크립트 예제

백업 스크립트의 예제

```
#!/bin/bash
#Script to backup the home directory

tar -cf backup-home.tar /home/ec2-user

echo "backup job complete at `date`"
```

스크립트 결과

```
[ec2-user]$ ./backup.sh
tar: Removing leading `/' from member names
tar: /home/ec2-user/backup-home.tar: file is the archive; not dumped
"backup job complete at Fri Jun 18 08:13:30 UTC 2021"
[ec2-user]$ ls
backup-home.tar  displayName.sh  guess.sh  scripts  whilebreak
```

셸 스크립트



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

```
echo "Hello $USER!"
```

```
echo "Today's date is : `date`"
```

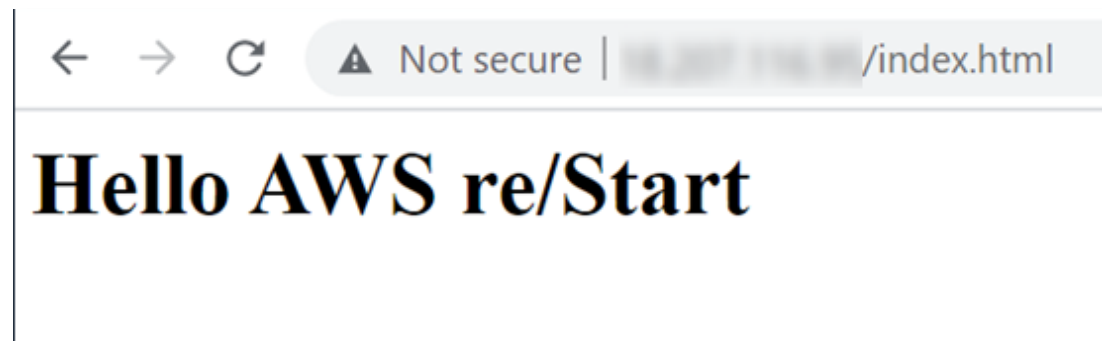
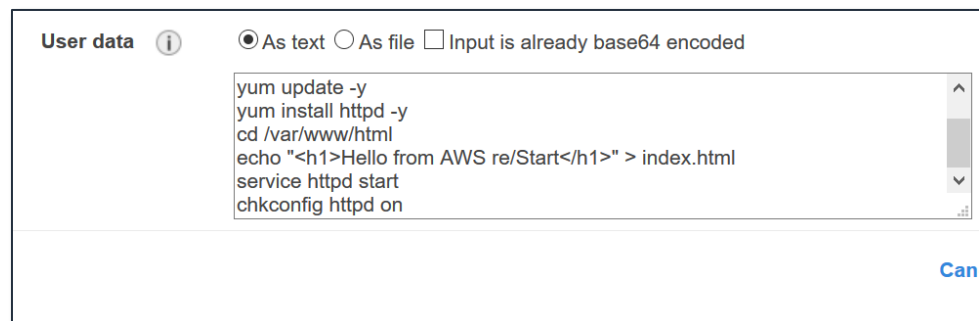
텍스트 편집기에
스크립트 쓰기

```
[ec2-user]$ ./hello.sh
Hello ec2-user
Today's date is : Fri Jun 18 08:22:27 UTC 2021
[ec2-user]$
```

스크립트 실행

Amazon EC2 사용자 데이터 스크립트

- Amazon Elastic Compute Cloud(Amazon EC2)는 가상 컴퓨팅 서비스입니다.
- 셸 스크립트는 생성 시 EC2 인스턴스에 소프트웨어를 설치할 수 있습니다.





기본 스크립팅 구문

문자

- Bash는 #으로 시작하는 줄을 무시합니다.
- # 문자는 사용자에게 지침이나 옵션을 제공할 수 있는 주석이나 메모를 정의하는데 사용됩니다.

```
[ec2-user]$ cat script.sh
#This is a comment
#echo "this line will be ignored"
echo "This line will print a message"
[ec2-user]$ ./script.sh
This line will print a message
[ec2-user]$
```

두 번째 echo 명령이 실행되고 메시지를 표시합니다.

#!/bin/bash 및 #comments

- #!는 shebang이라고 합니다.
- 첫 번째 줄은 사용할 인터프리터를 정의합니다(인터프리터 경로와 이름 제공).
- 스크립트는 셸이 스크립트를 실행할 지시문으로 시작해야 합니다.
- 위치와 셸은 다를 수 있습니다.
- 각 셀에는 예상되는 구문을 시스템에 알려주는 고유한 구문이 있습니다.

예: #!/bin/bash

- #으로 스크립트의 목적, 작성자 정보, 스크립트에 대한 특수 지시문, 예제 등을 포함한 주석을 정의합니다.

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

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

스크립트 설명서

- 일부 관리자는 모든 관련 정보와 섹션이 포함된 스크립트 템플릿을 만듭니다.
- 템플릿에 포함되는 내용은 다음과 같습니다.
 - 제목
 - 목적
 - 작성자 이름 및 연락처
 - 특별 지침 또는 예제

```
#!/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

#.....
#Script body
```

스크립트 설명서의 예제

Bash 변수 선언

- 고유한 변수를 선언(생성)합니다.
- 스크립트에서는 이러한 고유 변수를 사용합니다.

```
[ec2-user]$ cat displayName.sh
#!/bin/bash
NAME="Mary Major"
echo $NAME
[ec2-user]$ ./displayName.sh
Mary Major
[ec2-user]$
```

NAME이라는 변수가 생성되었습니다. 이 변수는 Mary Major라는 이름으로 설정됩니다.

유용한 명령

명령	설명
echo	콘솔에 정보 표시
read	사용자 입력 내용 읽기
subStr	문자열의 하위 문자열 가져오기
+	숫자 두 개를 더하거나 문자열 결합
file	파일 열기
mkdir	디렉터리 만들기
cp	파일 복사
mv	파일 이동 또는 이름 바꾸기
chmod	파일에 권한 설정
rm	파일, 폴더 등 삭제
ls	디렉터리 나열

데모: 스크립트에서 변수 선언



개요

- 일부 변수는 기본값으로 구성되지만, 고유한 변수를 선언할 수도 있습니다. 스크립트를 작성하고 실행할 때 이러한 기능이 자주 사용됩니다. 이 경우 변수를 선언하는 간단한 스크립트를 작성하고, 해당 변수를 실제로 사용합니다.
- 이 데모에서는 사용자 이름 저장 방법을 보여주는 간단한 Bash 스크립트를 만들어 봅니다.

연산자

=는 문자열에 변수를 할당하는 데 사용됩니다.

\$는 변수를 평가하는 데 사용됩니다.

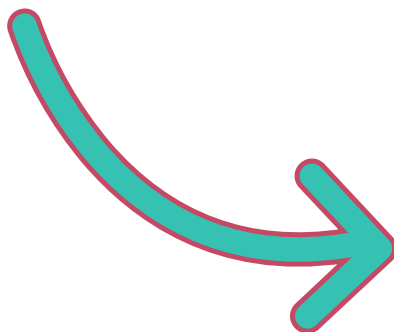
```
#!/bin/bash  
sum=$(( $1 + $2 ))  
echo $1 + $2 equals $sum
```

+는 수학 연산자로 사용됩니다.

표현식

표현식은 스크립트나 프로그램이 실행될 때 발생하는 질문에 답하는 방법입니다.

표현식으로 사용된
변수 `sum`



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

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

```
echo $1 + $2 equals $sum
```


조건 스테이트먼트

조건 스테이트먼트를 사용하면 테스트 성패에 따라 스크립트에서 다양한 액션을 수행할 수 있습니다.

다음 스크립트에서 `rm` 명령은 `cp` 명령이 성공적으로 완료됐을 때에만 실행됩니다.

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

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

조건

```
[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]$
```

결과: file1은 이제 존재하지 않습니다.



논리 제어 스테이트먼트

if 스테이트먼트

- 첫 번째 명령이 종료 코드 0(성공)으로 성공하면 후속 명령이 실행됩니다.
- 이는 가장 간단한 조건 스테이트먼트입니다.
- if 스테이트먼트는 fi 키워드로 끝나야 합니다.

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

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

if - else 스테이트먼트

- 두 가지 액션을 정의합니다.
 - 조건이 true인 경우(then)
 - 조건이 false인 경우(else)

else
스테이트먼트

```
#!/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
```

file1이 없으므로
else 스테이트먼트가 호출됩니다.

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

if - elif - else 스테이트먼트

- 세 가지 액션을 정의합니다.
 - 조건이 true인 경우(then)
 - 조건이 false이지만 다른 조건이 true인 경우(then)
 - 두 번째 조건이 false인 경우(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
```

test 명령

- 파일 유형을 확인하고 값을 비교합니다.
- 조건이 설정되고 true이면 0, false이면 1로 테스트가 종료됩니다.
- 구문: `test <EXPRESSION>`

비교 결과가 true인지,
false인지 확인합니다.



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

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

비교 결과가 false입니다.



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

정수 비교 연산자

비교 연산자는 두 변수 또는 수량을 비교합니다.

- `-eq`는 서로 같음: `if ["$a" -eq "$b"]`
- `-ne`는 서로 다름: `if ["$a" -ne "$b"]`
- `-gt`는 전자가 더 큼: `if ["$a" -gt "$b"]`
- `-ge`는 전자가 더 크거나 서로 같음: `if ["$a" -ge "$b"]`
- `-lt`는 전자가 더 작음: `if ["$a" -lt "$b"]`
- `-le`는 전자가 더 작거나 서로 같음: `if ["$a" -le "$b"]`
- `<` 전자가 더 작음(이중 괄호 내): `(("$a" < "$b"))`
- `<=` 전자가 더 작거나 서로 같음(이중 괄호 내): `(("$a" <= "$b"))`
- `>` 전자가 더 큼(이중 괄호 내): `(("$a" > "$b"))`
- `>=` 전자가 더 크거나 같음(이중 괄호 내): `(("$a" >= "$b"))`

정수 비교 연산자

비교 연산자는 두 변수 또는 수량을 비교합니다.
다음 테스트는 서로 등가입니다.

```
if [ "$1" -gt "$2" ];  
then  
    echo "..."  
fi
```

```
if ((" $1 > $2 "))  
then  
    echo "..."  
fi
```

```
if [[ $1 -gt $2 ]]  
then  
    echo "..."  
fi
```

```
if [ $1 -gt $2 ]  
then  
    echo "..."  
fi
```

```
if (($1 > $2));  
then  
    echo "..."  
fi
```


문자열 비교 작업

- = 또는 ==는 같음
 - `if ["$a" = "$b"]`
 - `if ["$a" == "$b"]`
- !=는 다름
 - `if ["$a" != "$b"]`
 - 이 연산자는 `[[...]]` 구성에서 패턴 매칭을 사용합니다.
- ASCII 알파벳순으로 <는 더 작음
 - `If [[" $a " < " $b "]]`
 - `If [" $a " \< " $b "]`
 - `[]` 구성에서 <는 반드시 이스케이핑(escaped)되어야 한다는 점에 유의합니다.
- ASCII 알파벳순으로 >는 더 큼
 - `if [["$a" > "$b"]]`
 - `if ["$a" \> "$b"]`
 - `[]` 구성에서 >는 반드시 이스케이핑(escaped)되어야 한다는 점에 유의합니다.
- -z 문자열은 null임(즉, 길이가 0)
- -n 문자열은 *null*이 아님

문자열 비교 작업의 예제

```
#!/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]$
```

출력

루프 스테이트먼트

- 스크립트 섹션이 반복되도록 구성할 수 있습니다.
- 루프 종료 조건
 - 특정 횟수 반복 후(`for` 스테이트먼트)
 - 조건 충족 후(`until` 스테이트먼트)
 - 조건이 `true`인 경우(`while` 스테이트먼트)
- 반복은 스크립트 기능과 복잡성을 확장합니다.



for 스테이트먼트

- 지정된 횟수만큼 명령 반복
- do 및 done으로 괄호로 묶음

```
#!/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]$
```

while 스테이트먼트

- 지정된 조건이 true인 동안 스크립트를 계속 실행
- while 및 done으로 괄호로 묶음

```
#!/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]$
```

until 스테이트먼트

- while 스테이트먼트와 유사
- 조건이 true가 될 때까지 코드 실행
- until 및 done으로 괄호로 묶음

```
#!/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]$
```

루프 제어 스테이트먼트

- 루프와 함께 사용하여 조건을 더 잘 관리합니다.
 - `break`: 전체 루프 실행을 중지합니다.
 - `continue`: 조건이 충족되면 루프에서 벗어납니다. 루프는 계속 실행됩니다.
- 예: 지정된 값이 발견되면 루프를 중단하고 스크립트를 계속 진행합니다.
- 중첩 루프에 사용되어 루프의 다음 부분으로 계속 이어집니다.

```
[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"
```

루프 제어 스테이트먼트

```
#!/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]$
```



true 및 false 명령

- true 및 false 명령은 루프와 함께 조건 관리에 사용됩니다.
- 이러한 명령은 미리 결정된 종료 상태(true 또는 false 상태)를 반환합니다.
- **부울 표현식**은 true 또는 false로 평가될 때 값을 생성하는 표현식으로 사용됩니다.
- 사용자가 개별 조건을 만들 수 있습니다.

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

while true
do
    echo "loops forever"
done
```

true 또는 false로
평가됨



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

counter=1
while [ $counter -le 10 ];
do
    echo $counter
    ((counter++))
done
```

exit 명령

- 이전 부분에서 실패하면 스크립트 실행을 멈추고 셸로 나갑니다.
- 테스트 시 유용합니다.
- 코드 상태를 반환할 수 있습니다. 각 코드는 특정 오류와 연관될 수 있습니다.
- 예를 들면 다음과 같습니다.
 - `exit 0`: 프로그램이 오류 없이 완료됨
 - `exit 1`: 프로그램에 오류가 있음
 - `exit n`: 프로그램에 특정 오류가 있음
- `$?`은 마지막으로 실행된 명령의 처리 상태를 가져오는 명령입니다.

```
#!/bin/bash

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

명령 대체(검토)

- 명령은 다른 명령 구문에 배치할 수 있습니다.
- 명령은 백틱(`)으로 둘러싸여 있습니다.
- 명령은 스크립트에서 유용할 수 있습니다.

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

```
echo "Hello $USER!"
```

```
echo "Today's date is : `date`"
```

← 명령 대체

인수

- 인수는 스크립트로 작동하려는 값입니다.
- 인수를 공백으로 구분된 스크립트 호출 명령에 포함하여 스크립트에 전달합니다.
- 예를 들어, \$1은 첫 번째 인수이고 \$2는 두 번째 인수입니다.

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

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

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

read 명령

사용자가 입력한 내용을 스크립트로 읽음

입력 내용을 변수로 설정

사용자가 입력한 내용을 변수로 설정

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

```
echo " Hello. what is your name ? "  
read VARNAME  
echo " Glad to meet you, $VARNAME"
```

변수를 호출하여 출력으로 사용

```
[ec2-user]$ ./welcome.sh  
Hello. What is your name ?  
joe  
Glad to meet you, joe  
[ec2-user]$
```



셀 스크립트 실행

스크립트 권한(검토)

- 스크립트 파일에는 반드시 실행 권한이 있어야 합니다.
- `chmod 744 hello.sh`는 사용자가 스크립트를 실행하도록 허용하지만, 그룹이나 다른 사용자는 허용하지 않습니다(절대 모드 사용).
- `chmod u+x hello.sh`는 사용자가 스크립트를 실행하도록 허용하지만, 그룹이나 다른 사용자는 허용하지 않습니다(상징 모드 사용).

```
[ec2-user]$ ls -al hello.sh
-rwxr-xr-x 1 ec2-user ec2-user 67 Jun 15 12:44 hello.sh
[ec2-user]$ chmod 744 hello.sh
[ec2-user]$ ls -al hello.sh
-rwxr--r-- 1 ec2-user ec2-user 67 Jun 15 12:44 hello.sh
[ec2-user]$ chmod u-w hello.sh
[ec2-user]$ ls -al hello.sh
-r-xr--r-- 1 ec2-user ec2-user 67 Jun 15 12:44 hello.sh
[ec2-user]$
```

`rwxr- -r- -`는 `r-xr- -r- -`가 됨

Bash에서 스크립트 실행

- Bash는 \$PATH 변수를 사용하여 실행 파일을 검색합니다. Bash는 실행 가능한 모든 명령이나 스크립트가 해당 경로를 따라 있을 것으로 가정합니다.
- 스크립트가 해당 경로를 따라 있지 않으면 실행 파일 이름 앞에 ./를 붙입니다.
- 현재 세션 이후에도 지속되도록 설정을 업데이트해야 합니다.
- 검토: 사용자 홈 디렉터리에 저장된 파일에서 사용자 프로필이 로드됩니다.
 - /home/username/.bashrc
 - /home/username/.bash_history
 - /home/username/.bash_profile
- 예를 들어, 홈 디렉터리에서 myscript.sh라는 스크립트를 작성하고 이를 실행하려고 합니다.
 - myscript.sh는 Bash가 홈 디렉터리에서 실행 파일을 확인하지 않기 때문에 실패합니다.
 - ./myscript.sh는 성공합니다.

스크립트 실행 및 ./(검토)

- Bash는 \$PATH를 따라 실행 파일을 확인하는데, 홈 디렉터리는 포함되지 않습니다(포함되면 안 됨).
- \$PATH에 저장되지 않은 스크립트를 실행하려면 스크립트 앞에 ./를 사용합니다.
- 예: ./hello.sh
 - 이 예제에서는 스크립트가 현재 디렉터리에 있는 것으로 가정합니다.
 - 그렇지 않은 경우 절대 경로, 상대 경로 또는 별칭(예: \$HOME)을 제공해야 합니다.
 - cron 작업에는 항상 전체 경로가 있어야 합니다.

```
[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]$
```

확인 질문



셀 스크립트를 사용하여 자동화할 수 있는 태스크는 무엇입니까?



조건 스테이트먼트와 함께 셀 스크립트를 사용할 수 있는 시나리오는 무엇입니까?

요점



- Bash Shell Script는 한 파일에 복잡한 다중 명령 작업을 자동화하는 방법을 제공합니다.
- 스크립트 템플릿을 사용하면 스크립트에 적절히 문서화할 수 있습니다.
- 스크립트를 실행하려면 적절한 권한이 필요합니다.
- 스크립트가 사용하는 변수는 명령줄에서 제공되거나, 스크립트를 실행하는 사용자로부터 대화식으로 제공될 수 있습니다.
- 조건 스테이트먼트는 스크립트에서 다른 논리 경로를 생성합니다(if, gt, equality 등 사용).
- **보안!** 스크립트에 필요한 기능이 있는지 확인합니다.
- **테스트!** 모든 스크립트가 예상대로 작동하는지 확인합니다.



감사합니다.

© 2021 Amazon Web Services, Inc. 또는 자회사. All rights reserved. 본 내용은 Amazon Web Services, Inc.의 사전 서면 허가 없이 전체 또는 일부를 복제하거나 재배포할 수 없습니다. 상업적인 복제, 임대 또는 판매는 금지됩니다. 수정해야 할 사항, 피드백 또는 기타 질문이 있다면 <https://support.aws.amazon.com/#/contacts/aws-training>에서 문의해 주십시오. 모든 상표는 해당 소유자의 자산입니다.

