Part 01. 리눅스 개발 환경

Chapter 06. 쉘 문법

진행 순서

Chapter 06_01 쉘 스크립트 개요
Chapter 06_02 쉘 변수
Chapter 06_03 쉘 의사결정
Chapter 06_04 쉘 함수

Chapter 06_01 쉘 스크립트 개요

쉘 스크립트(shell script)는 쉘이나 명령 줄 인터프리터에서 돌아가도록 작성되었거나 운영 체제를 위해 쓰인 스크립트이다. 단순한 도메인 고유 언어로 여기기도 한다. 쉘 스크립트가 수행하는 일반 기능으로는 파일 이용, 프로그램 실행, 문자열 출력 등이 있다. – Wikipedia

장점

쉘 스크립트는 다른 프로그래밍 언어의 같은 코드로 쓰인 것보다 훨씬 더 빠른 경우가 많다.
다른 해석 언어에 비해 쉘 스크립트는 컴파일 단계가 없기 때문에 디버깅을 하는 동안 빠르게 실행할 수 있다.

단점

스크립트 내에 많은 명령들이 수행될 경우 각 명령에 대한 새로운 프로세스의 필요에 따라 많은 프로세스들이 생성됨을 필요로 함으로 속도가 느려질 수 있다.
단순 쉘 스크립트는 다양한 종류의 유닉스, 리눅스, BSD 등 운영체제의 시스템 유틸리티와 잘 호환된다는 장점이 있지만 복잡한 쉘 스크립트의 경우 쉘, 유틸리티, 다른 필수 요소 간의 차이가 많은 경우 실패할 가능성이 있다. (각 운영체제가 제공하는 유틸리티 명령 등이 다를 경우 수행이 안될 수 있다.)

Chapter 06_02 쉘 변수

echo “Hello World”
“echo”는 쉘 프로그램에서 출력을 수행
“Hello World”라는 문자열을 표준출력으로 보냅니다.

shvar=“Hello World”
쉘을 통해 변수에 값을 저장할 수 있습니다.
문자열은 큰 따옴표로 묶어 변수가 전체 문자열을 나타낼 수 있도록 하고
“=“ 주위에 공백이 없습니다.

echo $shvar
쉘 변수의 값은 앞에 “$”를 붙여서 얻을 수 있습니다.
해당 쉘 변수에 값을 저장하지 않았다면 빈 줄이 생김

cp $olddir $newdir
쉘 변수에 저장된 값은 다른 프로그램의 매개 변수로도 사용할 수 있습니다.

$shvar=“”
널 문자열을 지정하여 쉘 변수에 저장된 값을 지울 수 있습니다.

Chapter 06_02 쉘 변수

mv $myfile $myfile2
“myfile”이라는 쉘 변수에 파일 이름이 있고 같은 이름을 가지고 있지만
“2”가 붙은 다른 파일로 해당 파일을 복사하려고 한다고 가정,
그러나 쉘은 “myfile2”가 다른 쉘 변수라고 생각하고 작동하지 않음

mv $myfile ${myfile}2
위와 같이 사용할 수 있습니다.

쉘 프로그램에서 다른 쉘 프로그램을 호출하고 호출 프로그램과 동일한 쉘 변수를 사용하게 하려면
다음과 같이 “export(내보내기)“ 해야 합니다.

```
#!/bin/sh
shvar=“Hello World”
export shvar
echo “Call shtest2”
./shtest2.sh
echo “Done”
```

shtest1.sh

```
#!/bin/sh
echo “This is shtest2”
echo $shvar
```

shtest2.sh

```
[root@localhost ch6]# ./shtest1.sh
Call shtest2
This is shtest2
Hello World
Done
```

Chapter 06_02 쉘 변수

$1
"$(숫자)"를 통해 쉘 프로그램 매개변수를 참조할 수 있다.
"$1"은 첫 번째 매개변수, "$2"는 두 번째 매개변수, ...

```
#!/bin/sh
first=$1
second=$2
echo "1- $1"
echo "2- $2"
```

shtest3.sh

```
[root@localhost ]# chmod +x ./shtest3.sh
[root@localhost ]# ./shtest3.sh Hello World
"1- Hello"
"2- World"
```

$#
쉘 프로그램 매개변수의 개수

$*
쉘 프로그램 매개변수 전체 내용($1, $2, ...)

$$
쉘 프로그램 실행 프로세스 ID

$!
쉘 프로그램이 실행시킨 백그라운드 프로세스 ID

$?
쉘 프로그램이 실행한 프로그램 종료값(리턴값)

shtest4.sh

```
#!/bin/sh

echo "argument number is $#"
echo "the argument $*"
echo "pid $$"

top &
echo "background pid $!"

ls
echo "result $?"
```

```
[root@localhost ]# chmod +x ./shtest4.sh
[root@localhost ]# ./shtest4.sh test shell
argument number is 2
the argument test shell
pid 2711
background pid 2712
top: failed tty get
shtest1.sh  shtest2.sh  shtest3.sh  shtest4.sh
result 0
```

Chapter 06_03 쉘 의사결정

쉘 프로그램은 인수와 변수에 대해 조건부 테스트를 수행하고 결과에 따라 다른 명령을 실행할 수 있습니다.

```
#!/bin/sh
if [ "$1" = "fork" ]
then
    echo "fork not allowed."
    exit
elif [ "$1" = "knife" ]
then
    echo "knife not allowed."
    exit
else
    echo "fork & knife not allowed"
    fi
echo "spoon please"
```

shtest5.sh

```
[root@localhost ]# chmod +x ./shtest5.sh
[root@localhost ]# ./shtest5.sh fork
fork not allowed.
[root@localhost ]# ./shtest5.sh knife
knife not allowed.
[root@localhost ]# ./shtest5.sh
fork & knife not allowed
spoon please
```

명령 행을 점검하여 첫 번째 인수가 “fork"또는 “knife"인지 확인하고 "exit"명령을 사용하여 종료합니다.
다른 인수를 사용하면 나머지 파일을 실행할 수 있습니다.
"$ 1"을 큰 따옴표로 묶는 방법에 유의하십시오.
따라서 테스트에서 널 결과가 발생하면 오류 메시지가 생성되지 않습니다.
if [$1 = “fork”] 와 같이 큰 따옴표를 쓰지 않았을 때 인자가 없다면
./shtest5.sh: line 2: [: =: unary operator expected (에러 발생)

Chapter 06_03 쉘 의사결정

문자열
[-n $shvar] – 문자열의 길이가 0보다 큰지
[-z $shvar] – 문자열의 길이가 0인지
[“$shvar” = “fox”] – 문자열이 같으면 true
[“$shvar” != “fox”] – 문자열이 다르면 true
[“$shvar” = “”] – 문자열이 null이면 true
[“$shvar” != “”] – 문자열이 null이 아니면 true

숫자
[“$nval” –eq 0] - 0과 같으면 true
[“$nval” –ge 0] - 0과 같거나 크면 true
[“$nval” –gt 0] – 0보다 크면 true
[“$nval” –le 0] - 0보다 작거나 같으면 true
[“$nval” –lt 0] - 0보다 작으면 true
[“$nval” –ne 0] - 0과 다르면 true

파일
[-d tmp] – tmp 가 디렉토리면 true
[-f tmp] – tmp가 파일이면 true
[-r tmp] – tmp가 읽기 가능하면 true
[-s tmp] – tmp의 사이즈가 0이 아니면 true
[-w tmp] – tmp가 쓰기 가능하면 true
[-x tmp] – tmp가 실행 가능하면 true

결합
[conditionA –a conditionB] – 조건문 A, B 모두 참인지, AND
[conditionA –o conditionB] – 조건문 A, B 중 하나라도 참인지, OR

Chapter 06_03 쉘 의사결정

c언어의 switch와 유사한 case 구문 존재

```
case "$1"
in
    “fork”) echo “fork not allowed.”
        exit;;
    “knife”) echo “knife not allowed.”
        exit;;
    *) echo “fork & knife not allowed";;
esac
```

문자열 ";;" 각 "case"절을 종료하는 데 사용됩니다.

Chapter 06_03 쉘 의사결정

C언어의 for 구문과 유사한 for 반복문

```
for variable in value1 value2 ...
do
   command
done
```

```
for nvar in 1 2 3 4 5
do
   echo $nvar
done
```

shtest6.sh

```
[root@localhost ]# ./shtest6.sh
1
2
3
4
5
```

C언어의 while 구문과 유사한 while 반복문

```
while [ condition ]
do
   command
done
```

```
#!/bin/sh
n=5
while [ "$n" -ne 0 ]
do
   echo $n
   n=`expr $n - 1`
done
```

shtest7.sh

```
[root@localhost ]# ./shtest7.sh
5
4
3
2
1
```

Chapter 06_04 쉘 함수

C언어의 함수와 유사

```
func_name()
{
    command
}
```

```
num=5

func1()
{
        echo "func1 process"
}

func2()
{
        func_val=0
        echo "func2 process"
        while [ $func_val -lt $1 ]
        do
                echo $func_val
                func_val=`expr $func_val + 1`
        done
}

func1
func2 $num
```

shtest8.sh

```
[root@localhost ]# ./shtest8.sh
func1 process
func2 process
0
1
2
3
4
```