



13장. 배시셀 프로그래밍[2]

❖ 학습목표

- 다양한 쉘 변수를 이해하고 활용하는 방법을 익힌다
- 사용자로부터 입력을 받아 스크립트 파일에서 처리하는 방법을 익힌다
- 다양한 연산자와 문자열 테스트, 파일 테스트를 활용하는 방법을 익힌다
- 조건문과 반복문의 사용 방법을 익힌다
- 스크립트의 실행 오류를 찾아 수정하는 방법을 익힌다

❖ 내용

- 쉘 스크립트
- 쉘 변수 사용하기
- 사용자로부터 입력받기
- 연산자
- 제어문
- 디버깅

기본출력문 : echo

- ❖ 화면에 원하는 문자열을 출력
- ❖ 문자열에 공백이 있을 경우 전체를 “ ”로 묶어준다
- ❖ 자동으로 줄 바꿈을 한다.
 - -n 옵션을 쓰면 줄 바꿈을 하지 않는다.
- ❖ 명령어를 실행하여 그 결과를 출력가능 : 역따옴표 (` `) 를 사용

```
#!/bin/bash  
echo `ls -a /etc`
```

- ❖ 변수를 출력할 때 마지막에 입력된 값을 출력한다.

사용자로부터 입력 받기 : read

- ❖ 사용자의 표준입력을 변수로 받아들이는 역할
- ❖ 쉘 내장 명령으로 키보드나 파일로부터 입력 처리

❖ 사용형식

형식	의미
read x	표준입력에서 한 행을 입력 받아 x에 저장
read first last	표준입력에서 한 행을 입력받아 첫번째 단어를 first에 저장하고 나머지 모두를 last에 저장
read -p prompt	prompt를 출력하고 입력을 기다린다. 입력된 값은 REPLY 변수에 저장

사용자로부터 입력 받기 - read

❖ 사용 예 : test_read

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

```
# 키보드 입력 처리를 테스트 하는 스크립트
```

```
read x
```

```
echo $x
```

```
# 아무 메시지 없이 사용자 입력을 기다림
```

```
# 사용자가 임의의 값을 입력하면 x에 저장
```

```
read x y
```

```
echo $x $y
```

```
# 첫 단어는 x, 나머지는 y에 저장
```

```
# x, y 값 출력
```

```
read -p "Input : "
```

```
echo "input : $REPLY"
```

```
# Input : 을 출력한 후 입력 기다림
```

```
# $REPLY에 자동 저장된 입력값 출력
```

연산자

- ❖ 프로그램에서 자료를 처리하는 방법
- ❖ 산술 연산자, 비교 연산자, 논리 연산자, 비트 연산자 제공
- ❖ 수치 연산자 사용시 let 또는 (()) 사용해야 함

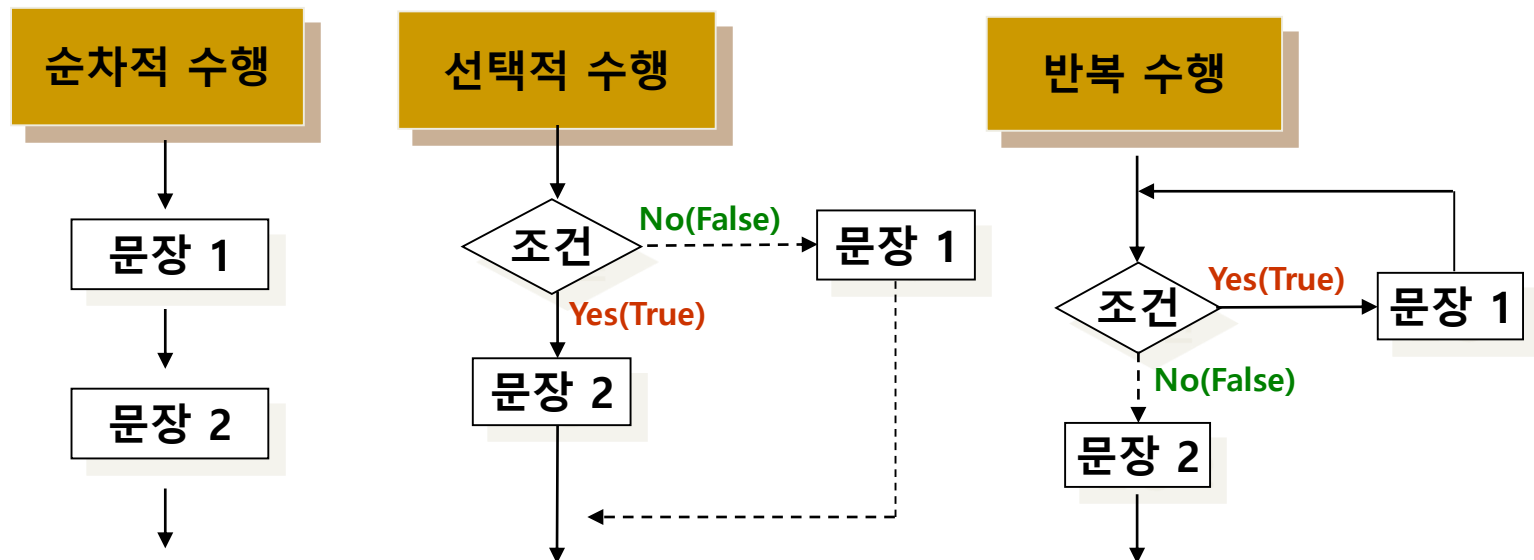
연산자	의미	사용예
-	음수 (단항연산)	-5
!	논리 부정(not)	((! x < y))
~	비트 반전 (not)	~y
* / %	곱셈, 나눗셈, 나머지 연산	let y=3 * 5
+ -	덧셈, 뺄셈	let x=x+1
<< >>	비트왼쪽 시프트, 비트오른쪽 시프트	((y = x << 3))
<= >= < > == !=	비교 연산	((x < y))
& ^	비트 AND, XOR, OR 연산	let "z = x ^ y"
&&	논리 AND, OR	((x<y x==3))
=	변수값 지정	let z=1
*= /= %= += == <<= >>= &= ^= =	단축 연산	let z+=1

❖ 사용 예

```
$ a=5
$ echo $a
5
$ let a = 20          ----> let에서 공백을 사용 못함
-bash: let: =: syntax error: operand expected (error token is "=")
$ let "a = 20"        ----> 공백을 포함하려면 " " 사용해야 함
$ print $a
20
$ (( a = 30 ))        ----> (( ))에서는 공백 사용 가능
$ echo $a
30
$ a=$a*5              ----> let이나 (( ))을 사용하지 않으면 문자열로 처리
$ echo $a
30*5
$ echo $((5*6))       ----> 계산 결과를 바로 출력할 수 있음
30
$ echo $(( ! 2 + 3 * 4 )) ----> 우선순위에 따라 ! 먼저 수행. 2는 0이 됨.
12
$ echo $(( 2 << 1 ))  ----> 왼쪽 shift는 *2와 같음. 2번 shift는 *4
4
$ echo $(( 3 ^ 5 ))   ----> XOR 연산 결과
6
$
```

05. 제어문

- ❖ 프로그램내의 문장 실행 순서를 제어하는 것
- ❖ 선택적 실행문
 - 프로그램 실행문을 조건에 따라 선택적으로 실행
 - if , select
- ❖ 반복 실행문
 - 프로그램 실행문을 정해진 횟수나 조건에 따라 반복 실행
 - while , do , for



선택적 실행문 - if~then~else

❖ 주어진 조건의 참, 거짓여부에 따라 명령 실행

```
if  조건명령
then
    명령
[ else
    명령 ]
fi
```

조건 명령을 실행하여

- 그 실행값이 0이 아닌 값이면 then다음의 명령을 실행하고,
- 0이면 else 다음의 명령을 실행한다.

❖ 예 : test_if

```
$ ./test_if
Input x : 100
Input y : 200
x is less than y
$
```

```
#!/bin/bash
# test_if : if 문을 테스트하는 스크립트
```

```
echo -n "Input x : "
read x          # x 값을 입력 받음
echo -n "Input y : "
read y          # y 값을 입력 받음
```

```
if (( x < y ))
then
    echo "x is less than y"
else
    echo "y is less than x"
fi
```

선택적 실행문 - if~then~elif ~ else

❖ 조건이 실패일 때 새로운 분기 명령 실행

```
if  조건명령1
then
    명령
elif 조건명령2
then
    명령
else
    명령
fi
```

❖ 예 : test_elif

```
$ ./test_elif
Input Score : 50
Your score is not good.
$
```

```
#!/bin/bash
# test_elif: if-elif 문 테스트

echo -n "Input Score : "
read score

if (( $score > 90 ))
then
    echo "Your score is great. "
elif (( $score >= 80 ))
then
    echo "Your score is good. "
else
    echo "Your score is not good. "
fi
```

선택적 실행문 - case 문

- ❖ 주어진 변수의 값에 따라 실행할 명령 따로 지정
- ❖ 변수의 값이 value1 이면 value1부터 ;;(또는 exit;;)을 만날 때까지 명령 실행
- ❖ case로 시작하여 esac로 끝난다.
- ❖ 일치하는 값이 없으면 기본값인 *에 설정된 명령을 실행

```
case 변수 in
value1)
    명령 ;;
value2)
    명령 ;;
*)
    명령 ;;
esac
```

선택적 실행문 - case 문

❖ 예 : case1.sh

```
#!/bin/bash
echo -n "What is your blood type? : "
read _blood

case "$_blood" in
    A)
        echo "Your blood type is A"
        exit;;
    B)
        echo "Your blood type is B"
        exit;;
    0)
        echo "Your blood type is 0"
        exit;;
    AB)
        echo "Your blood type is AB"
        exit;;
    *)
        echo "not correct blood type"
        exit;;
esac
```

실행결과

```
[root@localhost test]# sh case1.sh
What is your blood type? : A
Your blood type is A
[root@localhost test]# sh case1.sh
What is your blood type? : 1
not correct blood type
[root@localhost test]#
```

반복 실행문 - for

- ❖ 리스트 안의 각 값들에 대해 지정한 명령을 순차 실행

```
for 변수 in list
do
    명령
done
```

- ❖ 예 : test_for

```
#!/bin/bash
# test_for: for 테스트 스크립트

for num in 0 1 2 3 4 5
do
    echo number is $num
done
```

```
$ ./test_for
number is 0
number is 1
number is 2
number is 3
number is 4
number is 5
$
```

반복 실행문 - while

❖ 조건 명령이 정상 실행되는 동안 명령 반복

```
while 조건명령
do
    명령
done
```

❖ 예 : test_while

```
#!/bin/bash
# test_while: while을 이용해 1부터 10까지
# 합을 구하는 스크립트
```

```
count=1
sum=0
```

```
while (( count <= 10 ))
do
    (( sum += count ))
    let count+=1
done
```

```
print 1부터 10까지의 합 : $sum
```

```
$ ./test_while
1부터 10까지의 합 : 55
$
```

반복 실행문 - until

❖ 조건 명령이 정상 실행될 때까지 명령 반복

```
until 조건명령
do
    명령
done
```

```
$ ./test_until
로그인 이름 : hanb
hanb pts/3 Aug 19 01:07 (192.168.0.2)

$
```

❖ 예 : test_until

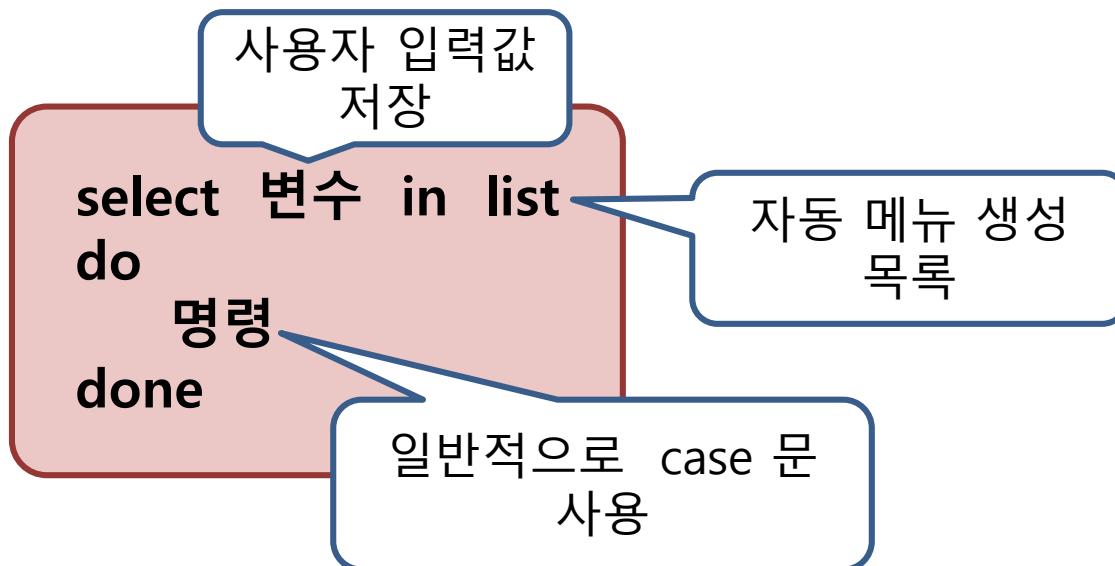
```
#!/bin/bash
# test_until: 지정한 사용자가 로그인하면 알리는 스크립트

echo -n "로그인 이름: " # 입력 안내문 출력
read person            # 유저 이름을 person에 저장

until who | grep $person # > /dev/null
do
    sleep 5              # 유저가 접속 중이 아니면 5초 쉬
done
echo "₩007"             # beep. 뽕 소리를 냄
```

반복 실행문 - select

- ❖ 메뉴를 생성할 수 있는 반복 실행문
- ❖ list에 지정한 항목을 자동으로 선택 가능한 메뉴로 만들어 화면에 출력해줌
- ❖ 사용자는 각 항목에 자동 부여된 번호를 선택
- ❖ 사용자 입력은 select와 in 사이에 지정된 변수에 저장
- ❖ 보통 case 문과 결합하여 입력 값 처리



반복 실행문 - select

❖ 예 : test_select

```
$ test_select
1) pwd
2) date
3) quit
명령을 입력하세요 : 1
/export/home/user1/unix/ch13
1) pwd
2) date
3) quit
명령을 입력하세요 : 3
$
```

```
#!/bin/bash
# test_select: 사용자 입력에 따라 pwd,date 명령실행

PS3="명령을 입력하세요 : "

select cmd in pwd date quit
do
    case $cmd in
        pwd)
            pwd ;;
        date)
            date;;
        quit)
            break;;
        *)
            echo 잘못 입력하셨습니다. 번호를 선택하세요.
            ;;
    esac
done
```

07. 디버깅

❖ 스크립트 실행도중 발생한 오류 수정 방법

❖ 구문 오류

- 셸이 실행도중 구문오류가 발생한 라인번호 출력

❖ 실행 오류

- 오류 메시지 없이 실행이 안되거나 비정상 종료

디버깅 : `bash -x`

- ❖ 가장 간단한 스크립트 실행 오류 수정방법
- ❖ 스크립트의 각 행이 실행될 때마다 화면에 출력됨

```
$ bash -x test_while
+ count=1
+ sum=0
+ (( count <= 10 ))
+ (( sum+=count ))
+ let count+=1
+ (( count <= 10 ))
+ (( sum+=count ))
+ let count+=1
+ (( count <= 10 ))
(중략)
+ print 1부터 10까지의 합 : 55
1부터 10까지의 합 : 55
$
```

스크립트의 내용

13장 요약 [1]

❖ 셸 스크립트 개요

- 셸이 제공하는 프로그래밍 구문과 유닉스 명령으로 구성된 텍스트 파일
- 실행방법 : `$ bash script` 또는 `$ chmod +x script` 하고 `$./script`

❖ 셸 변수 사용하기

- 변수 정의 : 변수명=값

❖ 사용자 입력 처리

- 키보드 입력 처리 : `read`

❖ 연산자

- 산술, 비교, 논리, 비트 연산자 제공
- 사용방법: `let` 다음 또는 `(())` 안에 연산식 기술

13장 요약 [2]

❖ 조건문

- if 문 : if 조건명령이 0이 아닌 값이면 then, 0이면 else를 실행
- case 문 : 변수의 값에 따라 선택적으로 명령 실행

❖ 반복문

- for 문 : 리스트에 있는 값을 순차적으로 변수에 저장하여 명령 실행
- while 문 : 조건 명령이 성공인 동안 명령 반복 실행
- until 문 : 조건 명령이 실패인 동안 명령 반복 실행
- select 문 : 선택 가능한 메뉴를 생성하는 반복문으로 선택에 따른 동작을 지정하려면 case 문과 함께 사용

❖ 디버깅

- 스크립트의 실행 오류 수정 방법
- bash -x script : 지정한 script를 한행씩 실행하며 결과를 화면에 출력