Contribution

Marathon

Innovation

엔터프라이즈 서버관리

13주차 : 셸 스크립트 2

2024년 1학기

목 차

1. 선택 구조

1. 선택 구조

제어 구조

- 셸 스크립트에서 실행을 제어하기 위해 선택과 반복 구조를 사용함
- if 명령, for 명령, case 명령, while 명령, until 명령 — 셸예약어
  - 선택 구조
  - 반복 구조

if 명령(1)

- 프로그래밍 언어에서 if 문의 기능을 수행하는 복합 명령
- 문법

```
if command...; then
        command...
[ elif command...; then
        command... ]...
[ else
        command...]
  fi
```

then과 구분을 위해 세미콜론(;)을 써야 함

여러 번 나올 수 있음

1. 선택 구조

if 명령(2)

- 세미콜론(;)은 같은 라인에서 다른 단어(then, elif, else, fi 등)와 구분이 필요할 때 사용함
- if 다음의 명령을 실행하여 참이면 then 다음의 명령을 실행함(if 명령은 종료됨)
  - if나 elif 다음에 조건 검사를 위한 test 명령을 사용할 수 있음
  - if나 elif 다음에 여러 명령이 나오면 마지막 명령의 종료 상탯값으로 참과 거짓을 구분함
  - 종료 상탯값 0은 성공적 종료를 의미하며 참으로 간주함 예) $?
- 거짓이면 elif 다음의 명령을 실행하여 참/ 거짓을 판단하고 실행함
- 만족되는 것이 없으면 else 다음의 명령을 실행함

1. 선택 구조

if 명령의 실행 예(1)

```
$ cd /usr/bin
$ echo $?
0    정상
$ cd /bin/usr
bash: cd: /bin/usr: 그런 파일이나 디렉터리가 없습니다
$ echo $?
1    오류
$ if true; then
> echo "Success"
> else
> echo "Failure"
> fi
Success
$ if true; false; then
> echo "True"
> fi
```

마지막 종료 상태값인 false가 반영

1. 선택 구조

```
[linux@localhost home]$ cd /usr/bin
[linux@localhost bin]$ echo $?
0
[linux@localhost bin]$ cd /bin/usr
bash: cd: /bin/usr: 그런 파일이나 디렉터리가 없습니다
[linux@localhost bin]$ echo $?
1
[linux@localhost bin]$ if true; then
> echo "Success"
> else
> echo "Failure"
> fi
Success
[linux@localhost bin]$ if false; then
> echo "Success"
> else
> echo "Failure"
> fi
Failure
[linux@localhost bin]$ if true; false; then
> echo "True"
> else
> echo "False"
> fi
False
[linux@localhost bin]$ if false; true; then
> echo "True"
> else
> echo "False"
> fi
True
```

1. 선택 구조

test 명령

- 조건 검사를 위해 사용하는 명령
  - 조건이 만족되면 종료 상탯값으로 0(참)을, 아니면 1(거짓)을 리턴함
- test *expression* 또는 [*expression*]
  - ***expression*** 은 파일의 상태 검사, 문자열의 비교, 정수 비교를 위한 조건 수식
  - 대괄호와 ***expression*** 사이에 공백이 있어야 함

```
$ if test -f .bashrc; then
> echo "It's a regular file."
> fi
It's a regular file.
$ if [ -f .bashrc ]; then
> echo "It's a regular file."
> fi
It's a regular file.
```

파일의 상태 검사를 위한 수식

```
[linux@localhost bin]$ if test -f.bashrc; then
> echo "It's a regular file."
> fi
It's a regular file.
[linux@localhost bin]$ if [ -f.bashrc ]; then
> echo "It's a regular file."
> fi
It's a regular file.
```

해당 파일이 존재하고 파일이 디렉터리가 아닌 정규파일이면 “True”

1. 선택 구조

test 명령에서 수식에 사용되는 연산자

	연산자	설명(아래 조건을 만족하면 참이다)	
파일의 상태 검사	file1 -nt file2	파일1이 파일2보다 새로운 것인가	newer than
	file1 -ot file2	파일1이 파일2보다 오래된 것인가	older than
	-d file	파일이 디렉터리인가	
	-e file	파일이 존재하는가	exist
	-f file	파일이 존재하고 정규 파일인가	
	-s file	파일이 존재하고 크기가 0보다 큰가(not empty)	
	-r file	파일이 읽기 가능한가	
문자열 비교	string1 > string2	문자열1이 문자열2보다 큰가(사전 순서). test 명령을 사용할 때는 비교를 위해 \>를 사용해야 함	
	string1 < string2	문자열1이 문자열2보다 작은가(사전 순서). test 명령을 사용할 때는 비교를 위해 \<를 사용해야 함	
	string1 = string2	문자열1이 문자열2과 같은가. ==를 사용할 수 있음	
	string1 != string2	문자열1이 문자열2과 다른가	
정수 비교	integer1 -eq integer2	정수1과 정수2가 같은가	equals to
	integer1 -ne integer2	정수1과 정수2가 다른가	not equals
	integer1 -le integer2	정수1이 정수2보다 작거나 같은가	less than or equals to
	integer1 -lt integer2	정수1이 정수2보다 작은가	less than

1. 선택 구조

if 명령의 실행 예

```
$ cat intCompare.sh
#!/bin/bash
if [ $# != 2 ]; then
  echo "You must supply two numbers as arguments"
  exit 1
fi
if [ $1 -eq $2 ]; then
  echo "$1 equals to $2."
elif [ $1 -gt $2 ]; then
  echo "$1 is greater than $2."
else
  echo "$1 is less than $2."
fi

echo "$1 + $2는 $[$1+$2]입니다."
$ chmod u+x intCompare.sh
$ ./intCompare.sh 36 68
36 is less than 68.
36 + 68는 104입니다.
```

인수의 개수가 2가 아니면

종료상태 값을 1로 함(비정상 종료)

수식 확장으로 덧셈을 수행

```
#! /bin/bash
if [ $# != 2 ]; then
        echo "You must supply two number as arguments"
        exit 1
fi

if [ $1 -eq $2 ]; then
        echo "$1 equals to $2."
elif [ $1 -gt $2 ]; then
        echo "$1 is greater than $2."
else
        echo "$1 is less than $2."
fi

echo "$1 + $2 = $[ $1+$2 ]"
```

```
[linux@localhost ~]$ ./intCompare.sh 36 36
36 equals to 36.
36 + 36 = 72
[linux@localhost ~]$ ./intCompare.sh 36 68
36 is less than 68.
36 + 68 = 104
```

1. 선택 구조

case 명령

elif를 여러 개 사용하는 것보다 가독성이 좋음

- 다중 선택을 지원하는 복합 명령
  - Java 언어에서 **switch** 문의 의미와 거의 같음
- 문법
  - **case** *word* **in**

    [*pattern* [| *pattern*]...) *command*...;;]...

    esac

    패턴을 끝낼 때는 오른쪽 괄호를 사용수행
- 설명
  - *word* 부분을 먼저 확장하고 *pattern*과 매칭되는지 검사함
  - 매칭이 이루어지면 상응하는 명령이 수행됨
  - 일단 매칭이 이루어지면 이후의 매칭 시도는 없음
  - *pattern* 에서 *는 프로그래밍 언어에서 'default' 키워드를 사용하는 것과 같음

    아무것도 매칭되는게 없을 때 수행

1. 선택 구조

case 명령에서 패턴의 사용 예

pattern)	설명(word 가 다음과 같은 경우 패턴 매칭이 일어남)
a)	a인 경우
[[:alpha:]])	1개의 알파벳 문자인 경우
???)	임의의 세 글자인 경우
*.txt)	.txt로 끝나는 경우
[aeiou])	모음에 해당하는 영문 소문자 1개인 경우
[ABE][0-9])	앞 글자가 A, B, E 중 하나이고 다음 글자가 숫자인 두 글자
*)	임의 길이의 글자와 매칭됨. case 명령에서 마지막 패턴으로 사용하는 것이 좋음

1. 선택 구조

case 명령의 실행 예

```
#!/bin/bash
clear    화면 클리어
echo "
Please Select:
a. Display System Information
b. Show Information about File Systems
c. Summarize Disk Usage Information
q. Quit
"
read -p "Enter selection [a, b, c, or q] > "
      prompt 의미
```

```
case $REPLY in
     a|A) echo "Hostname: $HOSTNAME"
          uptime  시스템의 사용정보 출력
          ;; break 의미
     b|B) df -h
          ;;
     c|C) if [ $(id -u) -eq 0 ]; then   UID(user id)가 0이면 root 사용자
            echo "All users' home disk Space utilization"
            du -sh /home/*
          else                  UID(user id)가 0이 아니면 일반 사용자
            echo "($USER)' home disk Space utilization"
            du -sh $HOME
          fi    디스크의 사용량 정보 출력
          ;;
     q|Q) echo "Program terminated."
          exit
          ;;
     *)   echo "Invalid entry" >&2
          exit 1  종료 상태 값을 1로 출력
          ;;
esac
```

1. 선택 구조

```
#! /bin/bash
clear
echo "
Please Select:
a. Display Syatem Information
b. Show Information about File Systems
c. Summarize Disk Usage Information
q. Quit
"

read -p "Enter selection [a, b, c, or q] > "

case $REPLY in
        a|A) echo "Hostname: $HOSTNAME"
                uptime
                ;;
        b|B) df -h
                ;;
        c|C) if [ $(id -u) -eq 0 ]; then
                echo "All users' home disk Space utilization"
                du -sh /home/*
        else
                echo "($USER)' home disk Space utilization"
                du -sh $HOME
              fi
              ;;
        q|Q) echo "Program terminated."
                exit
                ;;
        *) echo "Invalid entry" ?&2
                exit 1
                ;;
esac
```

```
Please Select:
a. Display Syatem Information
b. Show Information about File Systems
c. Summarize Disk Usage Information
q. Quit

Enter selection [a, b, c, or q] > a
Hostname: localhost
 14:50:44 up 14 min,  2 users,  load average: 0.04, 0.09, 0.08
```

```
Enter selection [a, b, c, or q] > b
Filesystem            Size  Used Avail Use% Mounted on
devtmpfs              4.0M     0  4.0M   0% /dev
tmpfs                 1.8G     0  1.8G   0% /dev/shm
tmpfs                 724M  9.7M  714M   2% /run
/dev/mapper/rl-root    36G  5.8G   30G  17% /
/dev/nvme0n1p1        960M  301M  660M  32% /boot
tmpfs                 362M  100K  362M   1% /run/user/1000
```

```
Enter selection [a, b, c, or q] > c
(linux)' home disk Space utilization
59M     /home/linux
```

```
Enter selection [a, b, c, or q] > q
Program terminated.
[linux@localhost ~]$ echo $?
0
```

```
Enter selection [a, b, c, or q] > z
Invalid entry ?
./inhatc.sh: 줄 30: 2: 명령어를 찾을 수 없음
[linux@localhost ~]$ echo $?
1
```

2. 반복 구조

2. 반복 구조

for 명령(1)

- 모든 데이터를 한 차례씩 처리하는 제어구조
- for *variable* [in *word*...]; do
  command...
  done
  종료부분
  시작부분
- 설명
  - ***word*** ... 부분(값의 목록)을 먼저 확장함
  - ***word*** ... 에 존재하는 값을 순차적으로 변수 ***variable*** 에 대입하고 **do**와 **done** 사이의 명령을 수행함
  - ***word*** 부분이 없다면 in "$@"가 있는 것으로 가정함
    인수가 한 개씩 대입됨

2. 반복 구조

for 명령의 실행 예(1)

```
$ for i in Kim Lee Park; do echo $i; done
Kim
Lee
Park
$ echo {A..D}
A B C D
$ for i in {A..D}; do echo $i; done
A
B
C
D
```

```
$ cat testFor.sh
#!/bin/bash
for FILE
do
    $FILE
done
$ chmod u+x testFor.sh
$ ./testFor.sh `ls`
testFor.sh
<중간 생략>
템플릿
```

```
[linux@localhost ~]$ for i in Kim Lee Park; do echo $i; done
Kim
Lee
Park
[linux@localhost ~]$ echo {A..D}
A B C D
[linux@localhost ~]$ for i in {A..D}; do echo $i; done
A
B
C
D
```

```
[linux@localhost ~]$ cat testFor.sh
#! /bin/bash
for FILE
do
    $FILE
done

[linux@localhost ~]$ chmod u+x testFor.sh
[linux@localhost ~]$ ./testFor.sh 'ls test*'
testFor.sh
```

2. 반복 구조

for 명령(2)

- C나 Java 프로그래밍에서 사용하는 형태
- for ((*exp1*; *exp2*; *exp3*)); do
  command...
  done

```
LIMIT=10
for ((a=0; a<LIMIT; a++)); do
  echo "$a"
done
```

- 설명
  - ***exp1*~*exp3*** 은 수식으로 어떤 것도 생략이 가능함
  - ***exp1***는 제어 변수의 초기화를 위해 한 번 수행됨
  - ***exp2***가 참인 동안 "명령과 ***exp3***"이 반복 수행됨

2. 반복 구조

for 명령의 실행 예(2)

- C나 Java 프로그램과 유사한 형태

```
$ cat testFor3.sh
#!/bin/bash

LIMIT=10
for ((a=0; a<LIMIT; a++)); do
  echo "$a"
done
$ .testFor3.sh
0
1
2
<중간 생략>
8
9
```

```
[linux@localhost ~]$ cat testFor3.sh
#! /bin/bash
LIMIT=10
for ((a=0; a<LIMIT; a++)); do
        echo "$a"
done

[linux@localhost ~]$ . testFor3.sh
0
1
2
3
4
5
6
7
8
9
```

2. 반복 구조

while 명령

- 조건이 참인 동안 명령을 반복하여 수행함
- while *command* ...; do

  command ...

  done
- 설명
  - **while** 다음에 나오는 명령을 실행하여 참 또는 거짓을 판단함
  - **if** 명령과 마찬가지로 종료 상탯값이 **0**이면 참으로 판단
  - 조건 비교를 위해 **while** 다음에 ***test expression*** 또는 ***[expression]*** 을 자주 사용함

2. 반복 구조

while 명령의 실행 예(1)

```
$ cat testWhile.sh
#!/bin/bash
N=1
S=0
while [ $N -le 10 ]; do
    echo -n "$N "
    S=$[$S+$N]  # S=$[S+N]
    N=$[$N+1]   # N=$[N+1]
done            # (( S=S+N ))
echo            # (( N=N+1 ))
echo $S
$ . testWhile.sh
1 2 3 4 5 6 7 8 9 10
55
```

test 명령

-n : 줄바꿈 않함

수식 확장

줄바꿈

```
[linux@localhost ~]$ cat testWhile.sh
#! /bin/bash
N=1
S=0
while [ $N -le 10 ]; do
        echo -n "$N "
        S=$[ $S+$N ] # S=$[ S+N ]
        N=$[ $N+1 ]  # N=$[ N+1 ]
done
echo
echo $S

[linux@localhost ~]$ . testWhile.sh
1 2 3 4 5 6 7 8 9 10
55
```

2. 반복 구조

for 명령과 while 명령

- C언어 형식의 for 명령(2)를 다음과 같이 변환할 수 있음
- ((*exp1*))

```
while (( exp2 )); do
    command...
    (( exp3 ))
done
```

- 설명
  - **((*expression*))**은 수식 계산에 사용되는 복합 명령
  - let "***expression***"과 동일하며 **test** 명령 대신에 사용할 수 있음
    - 조건 비교를 위해 **test *expression*** 또는 **[*expression*]** 와 함께 자주 사용함

2. 반복 구조

while 명령의 실행 예(2)

```
$ cat testWhile2.sh
#!/bin/bash

LIMIT=10
((a=0))
while (( a<LIMIT )); do
  echo "$a"
  (( a++ ))
done
$ ./testWhile2.sh
0
1
<중간 생략>
8
9
```

```
[linux@localhost ~]$ cat testWhile2.sh
#! /bin/bash
LIMIT=10
((a=0))
while (( a<LIMIT )); do
        echo "$a"
        (( a++ ))
done

[linux@localhost ~]$ chmod u+x testWhile2.sh
[linux@localhost ~]$ ./testWhile2.sh
0
1
2
3
4
5
6
7
8
9
```

2. 반복 구조

until 명령

- 조건이 만족될 때까지(거짓인 동안) 명령을 반복하여 수행함
- until *command* ...; do

  command ...

  done
- 설명
  - **until** 다음에 나오는 명령이 참이 될 때까지 반복함
    - 즉, 거짓인 동안 반복하게 됨
  - 조건을 표시하기 위해 **test *expression*, [*expression*]** 또는 **((*expression*))**을 사용할 수 있음

2. 반복 구조

until 명령의 실행 예

```
$ cat testUntil.sh
#!/bin/bash
N=1
S=0
until [ $N -gt 10 ]; do
    echo -n "$N "
    let S=$S+$N    # let S=S+N
    let N=$N+1     # let N=N+1
done               # (( S=S+N ))
echo               # (( N=N+1 ))
echo $S
$ . testUntil.sh
1 2 3 4 5 6 7 8 9 10
55
```

test 명령

```
[linux@localhost ~]$ cat testUntil.sh
#! /bin/bash
N=1
S=0
until [ $N -gt 10 ]; do
        echo -n "$N "
        let S=$S+$N     # let S=S+N
        let N=$N+1      # let N=N+1
done
echo
echo $S

[linux@localhost ~]$ . testUntil.sh
1 2 3 4 5 6 7 8 9 10
55
```

2. 반복 구조

While 명령

```
[linux@localhost ~]$ cat testWhile.sh
#! /bin/bash
N=1
S=0
while [ $N -le 10 ]; do
        echo -n "$N "
        S=$[ $S+$N ] # S=$[ S+N ]
        N=$[ $N+1 ]  # N=$[ N+1 ]
done
echo
echo $S

[linux@localhost ~]$ . testWhile.sh
1 2 3 4 5 6 7 8 9 10
55
```

Until 명령

```
[linux@localhost ~]$ cat testUntil.sh
#! /bin/bash
N=1
S=0
until [ $N -gt 10 ]; do
        echo -n "$N "
        let S=$S+$N    # let S=S+N
        let N=$N+1     # let N=N+1
done
echo
echo $S

[linux@localhost ~]$ . testUntil.sh
1 2 3 4 5 6 7 8 9 10
55
```

Contribution

Marathon

Innovation

The End !