쉘

리눅스 쉘

- $: 쉘 변수 (아래와 같은 예시처럼 변수가 무엇인지 기억이 나지 않을 때 많이 사용한다.)

```
i=100

echo "i:$i"
```

(test.sh 파일)

- 쉘 : 운영 체제 상에서 다양한 운영 체제 기능과 서비스를 구현하는 인터페이스를 제공하는 프로그램이다. 사용자와 커널 사이의 인터페이스를 감싸는 층으로 일반적으로 명령 줄(CLI)와 그래픽 형 (GUI) 2 종류로 분류된다.
- 쉘 스크립트 : 쉘로 하는 프로그래밍이다. 자바나 C언어처럼 조건문, 반복문 등의 일반적인 프로그래밍 문법을 사용할 수 있다. 그러나 컴파일 방식이 아닌 인터프리터를 통한 스크립트 방식이라는 차이점이 있다.

컴파일 언어	스크립트 언어
컴파일러를 거쳐서 기계어로 번역한 후 한 번에 실행 (번역 후 실행)	인터프리터가 코드를 한줄한줄 해석하며 실행 (번역하며 실행)
컴파일을 하기 위해 변수 선언 등의 제약사항이 많음	변수를 변수라고 선언하지도 않으며 해당 변수 자료형은 소스코드가 실행되는 순간임
코드 문법 등의 버그가 있으면 컴파일 자체가 안 됨	코드 실행 전까지 버그를 인지할 수 없기 때문에 오류 발견시점이 늦어짐 (디버깅이 까다롭다)
Java, C, C++ 등	Java Script, Python 등

- 쉘 스크립트를 사용하는 경우 : 서버가 다운되어서 확인해보니 서버 저장공간이 하나도 남아 있지 않았다. 이유는 로그 파일이 많이 쌓였기 때문이었다. 로그 파일이 업데이트가 안되어 관련 프로그램이 비정상적으로 종료되었다. 어떻게 하면 자동으로 오래된, 혹은 일정 시간 경과한 로그 파일을 삭제할 수 있을까? 이런 문제를 간단한 쉘 스크립트 생성 및 주기적 실행으로 해결할 수 있다.
- 쉘 스크립트 기본 문법
  1. 쉘 스크립트는 파일로 작성 후, 파일을 실행한다.
  2. 파일의 가장 위 첫 라인은 #!/bin/bash로 시작한다.
  3. 쉘 스크립트 파일은 코드를 작성한 후에는 실행 권한을 부여해야한다.
  4. 일반적으로 파일이름.sh 와 같은 형태로 파일 이름을 작성한다.
  5. 주석은 # 으로 처리한다.

echo

- ex1. bash 쉘에서 제공하는 echo 함수를 이용하여 화면에 "Hello bash!"를 출력할 수 있도록 스크립트를 작성해보자 (echo는 화면에 출력해주는 쉘 명령이다.)

```
[testuser@lms sh]$ vi hello.sh
[testuser@lms sh]$ ls -l
합 계  4
-rw-rw-r--. 1 testuser testuser 31  8월  22 18:05 hello.sh
[testuser@lms sh]$ chmod 777 hello.sh
```

```
#!/bi/bash

echo "Hello Bash!"
```

vi로 파일을 생성한 후 chmod로 권한을 부여한다.

```
[testuser@lms sh]$ sh hello.sh
Hello Bash!
```

./와 sh명령어로 실행할 수 있다.

echo -n : 마지막에 따라오는 개행 문자(new line)를 출력하지 않는다.

*개행문자 : 컴퓨터에서 줄바꿈을 나타내는 제어문자

변수

- ◆ 선언 : 변수명=데이터 (변수명=데이터 사이에 띄어쓰기는 허용되지 않는다.)
- ◆ 사용 : $변수명 (echo $변수명$변수명 ....)
- ◆ 코드 예시

```
[testuser@lms sh]$ vi variable.sh
```

```
#!/bin/bash

mysql_id='root'
mysql_directiory='/etc/mysql'

echo $mysql_id
echo $mysql_directiory
```

```
[testuser@lms sh]$ sh variable.sh
root
/etc/mysql
```

- ex2. 아이디 관련 정보 변수를 만들어보자 (이름, 나이, 직업)+

```
[testuser@lms sh]$ vi id.sh
[testuser@lms sh]$ sh id.sh
minsung20student
```

```
#!/bin/bash

name='minsung'
age=20
carrer='student'

echo $name$age$carrer
```

리스트 변수 (배열)

- 선언 : 변수명=(데이터 1 데이터 2 데이터 3)
- 사용 : ${변수명[인덱스번호]} : 인덱스 번호는 0 이 시작이다.
- 코드 예시

```
#!/bin/bash

a=("a" "b" "c") # 변수 a, b, c 선언

echo ${a[1]} # $a 배열의 두 번째 인덱스에 해당하는 b 출력
echo ${a[@]} # $a 배열의 모든 데이터 출력
echo ${a[*]} # $a 배열의 모든 데이터 출력
echo ${#a[@]} # $a 배열의 배열 크기 출력

filelist=($(ls)) # 해당 쉘 스크립트 실행 디렉토리의 파일 리스트를를 배열로 변수
echo ${filelist[*]} # $filelist 모든 데이터 출력
```

```
b
a b c
a b c
3
array.sh hello.sh id.sh tt.sh variable.sh
```

- 예제 3 : 아이디 관련 정보 리스트 변수로 만들고, 각 정보를 출력해보자 (이름, 나이, 직업)

```
#!/bin/bash

id=("leeminsung" 20 "student")
echo ${id[*]} # $id의 모든 데이터 출력
```

```
leeminsung 20 student
```

쉘 스크립트의 기본 문법은 까다롭다. 가변적으로 문법을 바꿔도 동작을 하긴 하지만 환경에 따라 동작을 달리한다.

- 사전에 정의된 지역 변수

  $$: 쉘의 프로세스 번호 (pid, 프로세스 아이디)

  $0 : 쉘 스크립트 이름

  $1~$9 : 명령줄 인수

  $* : 모든 명령줄 인수 리스트

$# : 인수의 개수
$? : 최근 실행한 명령어의 종료 값
0 성공, 1 ~ 125 에러, 126 파일이 실행 가능하지 않음, 128 ~ 255 시그널 발생

ls -al -z의 경우
ls가 $$, -al이 $1 , -z가 $2, $#은 2 개

- 예제 4 : 쉘 프로세스 번호, 쉘 스크립트 이름, 명령줄 인수, 모든 명령줄 인수 리스트, 인수 개수 출력해보기

```
#!/bin/bash

echo $$ $0 $1 $* $#
```

```
[testuser@lms sh]$ sh shell.sh
9433 shell.sh 0
[testuser@lms sh]$ sh shell.sh kkk
9462 shell.sh kkk kkk 1
[testuser@lms sh]$ sh shell.sh aaa bbb ccc ddd
9489 shell.sh aaa aaa bbb ccc ddd 4
```

연산자

- expr : 숫자 계산
  1. expr를 사용하는 경우 역 작은 따옴표(')를 사용해야한다.
  2. 연산자 *와 괄호 ( ) 앞에는 역 슬래시를 같이 사용해야한다. ~~wwwwww~~(붙여 쓴다)
  3. 연산자와 숫자, 변수, 기호 사이에는 space를 넣어야한다.

- 예제 5 : expr 명령으로 (30 / 5) - 8 계산해보기

```
#!/bin/bash

num=`expr \( 30 / 5 \) - 8`

echo $num
```

조건문

- 기본 if 문법 : then과 fi 안에만 들어가 있으면 되기 때문에 명령문을 꼭 탭으로 띄워야 하는 것은 아니다.

```
if [조건]
then
    명령문
fi
```

- 예제 6 : 두 인자값을 받아서 두 인자값이 다르면 'different values'를 출력해보자

```
#!/bin/bash

if [$1 != $2] # != 둘 이  다 르 면  이 라 는  의 미
then
   echo "different values"
   exit
fi
```

```
[testuser@lms sh]$ sh if.sh
different values
```

- 조건

  문자 1 == 문자 2 : 문자 1 과 문자 2 가 일치

  문자 1 != 문자 2 : 문자 1 과 문자 2 가 일치 X

  -z 문자 : 문자가 null이면 참 (값이 없으면 true)

  -n 문자 : 문자가 null이 아니면 참

- 수치비교 : <, >는 if 조건시 [[]]를 넣는 경우 정상 작동하기는 하지만, 기본적으로 다음 문법을 사용한다.

  값 1 -eq 값 2 : 값이 같음 (equal)

  값 1 -ne 값 2 : 값이 같지 않음 (not equal)

  값 1 -lt 값 2 : 값 1 이 값 2 보다 작음 (less than)

  값 1 -le 값 2 : 값 1 이 값 2 보다 작거나 같음 (less or equal)

  값 1 -gt 값 2 : 값 1 이 값 2 보다 큼 (greater than)

  값 1 -ge 값 2 : 값 1 이 값 2 보다 크거나 같음 (greater or equal)

- 파일 검사

  -e 파일명 : 파일이 존재하면 참

  -d 파일명 : 파일이 디렉토리면 참

  -h 파일명 : 파일이 심볼릭 링크 파일이면 참

  -f 파일명 : 파일이 일반파일이면 참

  -r 파일명 : 파일이 읽기 가능이면 참

  -s 파일명 : 파일크기가 0 이 아니면 참

  -u 파일명 : 파일이 set-user-id가 설정되면 참

  -w 파일명 : 파일이 쓰기 가능이면 참

  -x 파일명 : 파일이 실행 가능이면 참

- 예제 7 : 쉘 스크립트로 해당 파일이 있는지 없는지 확인해보자

```
#!/bin/bash

if [ -e $1 ]
then
  echo "file exist"
fi
```

```
[testuser@lms sh]$ sh if2.sh cal.sh
file exist
```

- 논리 연산
  조건 1 -a 조건 2 //AND
  조건 1 -o 조건 2 //or
  조건 1 && 조건 2 //양쪽 다 성립
  조건 1 || 조건 2 //한쪽 또는 양쪽 다 성립
  !조건 //조건이 성립하지 않음
  true //조건이 언제나 성립
  false //조건이 언제나 성립하지 않음
- 기본 if/else 구문
  if [조건]
  then
    명령문
  else
    명령문
  fi

- 예제 8 : 두 인자 값을 받아서 두 인자 값이 같으면 'same value', 다르면 'different value'가 출력되도록 해보자

```
#!/bin/bash

if [ $1 -eq $2 ]
then
   echo "same values"
else
   echo "different values"
fi
[testuser@lms sh]$ sh if3.sh 3 3
same values
```

- ping : 서버에서 여러 가지 컴퓨터가 연결되어 있을 때 연결된 특정 컴퓨터가 정상적으로 동작하는 지, 꺼져 있는지, 비정상적으로 동작하는지 확인하는 명령어이다. 해당 컴퓨터의 ip주소로 ping명령어를 실행한다. 그 주소에 확인 요청을 한다. 정상적인 컴퓨터는 응답을 한다. 응답이 오지 않으면 정상적으로 동작하지 않는다고 판단한다.

ping -c 1 192.168.0.105 1> /dev/null

0 : 표준 입력, 1 : 표준 출력, 2 : 표준 에러

1>/dev/null : 표준 출력 내용은 버려라 (출력하지 말아라)

-c 1 : 1 번만 체크해보라는 의미

- 코드 예제

```
#!/bin/bash

ping -c 1 192.168.0.105 1> /dev/null
if [ $? == 0 ] # $? : 가장 최근에 쉘 스크립트에서 실행한 명령의 결과값
then
   echo "Gateway ping success!" # 0일 경우 응답이 온 것이라 성공!
else
   echo "Gateway ping failed!" # 응답이 없을 때 나타남
fi
```

```
[testuser@lms sh]$ sh ping.sh
Gateway ping success!
```

- 조건문 한 줄에 작성하기 (if 구문)

`if [조건]; then 명령문; fi`

`if [ -z $1 ]; then echo 'Insert arguments'; fi`

if [뒤와] 앞에는 반드시 공백이 있어야함 [ ]에서 &&, ||, <, > 연산자들이 에러가 나는 경우는 [[ ]]를 사용하면 정상 작동하는 경우가 있음

반복문

- 기본 for 구분

for 변수 in 변수값 1 변수값 2 ....

do

  명령문

done

- 예제 8 : 현재 디렉토리에 있는 파일과 디렉토리를 출력해보자

```
for database in $(ls);
do
   echo $database
done
```

```
for database in $(ls); do
   echo $database
done
```

```
for database in $(ls); do echo $database; done
```

```
lists=$(ls)
num=${#lists[@]}
index=0
while [ $num -ge 0 ]
do
   echo ${lists[$index]}
   index=` expr $index + 1 `
   num=` expr $num - 1 `
done
```

- 기본 while 구문
  while [조건문]
  do
    명령문
  done

read : 사용자로부터 입력을 받기 위한 명령어

```
#!/bin/bash

echo "당 신 의  나 이 를  입 력 하 세 요 "
read age
echo "당 신 의  키 를  입 력 하 세 요 "
read height

echo "당 신 의  나 이 는  $age 키 는  $height 입 니 다 ."
```

```
당 신 의  나 이 를  입 력 하 세 요
20
당 신 의  키 를  입 력 하 세 요
188
당 신 의  나 이 는  20 키 는  188 입 니 다 .
```

declare : 변수 값을 선언한다.

declare -a : 각 이름을 배열 변수로 선언한다.

exec : 주어진 명령어를 실행하는데 새로운 프로세스를 생성하지 않고, 쉘 프로세스를 대체한다.