일일 업무 사항 정리

작성자	제품팀 이민성 인턴
업무 일시	2022.08.26

세부 사항

1. 업무 내역 요약 정리

목표 내역	Done & Plan
1. 쉘 스크립트 언어 이해 및 활용 2. 미로게임 만들기 3. 오라클 데이터베이스 설치과정 복기	오늘은 지금까지 공부했던 쉘 스크립트 언어에 대해 다시 한번 정리하고 상기시켰고 중간중간 새로운 개념을 공부했습니다. 기존에 배웠었지만 까먹었던 내용들과 헷갈리던 내용들이 정리되었습니다. 특히 헷갈렸던 if문에서 사용하는 특수기호나 지역변수 등을 조금 더 확실하게 알 것 같습니다. 그리고 미로 찾기에 대해 어떻게 구현할 지 생각해보고 코드를 짜봤는데 에러가 많이 발생하여서 다시 더 깊게 고민해보고 짤 생각입니다. 그리고 9 월 12 일까지 쉘 스크립트 언어에 대한 부족한 개념을 더 채우면서 미로 찾기를 개발할 계획입니다. - 미로 찾기 read를 사용해서 W,A,S,D로 방향키를 만듭니다. if else문과 echo, clear, sleep을 이용하여 깔끔하게 만들면 좋을 것 같습니다. 하지만 이 방식으로 하면 코드가 굉장히 길어질 뿐 아니라 복잡해질 것 같습니다. 일단은 이러한 방식으로 만들어보고 다듬어가며 다른 방법을 생각해서 만들어보면 좋을 것 같습니다.

2. 내용 세부 (업무 세부 내역 정리 및 기타 사항 정리)

쉘

1. 쉘이란?

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.

2. 쉘 스크립트란?

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 안됨	코드 실행 전까지 버그를 인지할 수 없기 때문에 오류 발견시점이 늦어짐 (디버깅이 까다롭다)
JAVA, C, C++ 등	Java Script, Python 등

3. 쉘 스크립트는 언제 사용할까?

서버가 다운되어서 확인해보니 서버 저장공간이 하나도 남아 있지 않았다. 그 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.

4. 쉘 스크립트의 기본 문법

- 쉘 스크립트는 파일로 작성 후, 파일을 실행한다.
- 파일의 가장 위 첫 라인은 #!/bin/bash로 시작한다.
- 쉘 스크립트 파일은 코드를 작성한 후에는 실행 권한을 부여해야 한다.
- 일반적으로 파일이름.sh 와 같은 형태로 파일 이름을 작성한다.
- 주석은 #으로 처리한다.

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

5. echo

echo는 화면에 출력해주는 쉘 명령이다.

```
#!/bi/bash

echo "Hello Bash!"
```

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

6. 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
```

```
[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
```

7. 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 개이다.

- 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
```

8. 환경변수 설정

- env: 전역 변수 설정 및 조회
- set: 사용자 환경 변수 설정 및 조회
- export: 사용자 환경 변수를 전역 변수로 설정

```
[testuser@lms sh4]$ # param1을 선언하고 전역 변수(env)와 사용자 환경 변수 (set)에서 확인
[testuser@lms sh4]$ # 사용자 환경 변수 (set)에만 값이 존재
[testuser@lms sh4]$ param1=Hello
[testuser@lms sh4]$ env | grep param1
[testuser@lms sh4]$ set | grep param1
param1=Hello
[testuser@lms sh4]$
[testuser@lms sh4]$ # param1을 export로 전역 변수로 변경
[testuser@lms sh4]$ export param1
[testuser@lms sh4]$ # 전역 전수(env)와 사용자 환경 변수 (set) 모두 존재
[testuser@lms sh4]$ env | grep param1
param1=Hello
[testuser@lms sh4]$ set | grep param1
param1=Hello
```

환경변수 설정은 현재의 세션에만 유효하다. 모든 세션에 적용하기 위해서는 .bashrc나 .profile같은 설정 파일에 선언해야 한다.

*사용자 환경 변수: 변수를 등록한 사용자만 사용할 수 있는 변수

*전역 변수: 모든 사용자가 같이 사용할 수 있는 변수

- set 명령어 옵션

-a	생성, 변경되는 변수를 export함
-e	오류가 발생하면 스크립트 종료
-x	수행하는 명령어를 출력 후 실행
-c	다음의 명령을 실행 ex) bash -c "echo 'A'".bash -c date
-o	옵션 설정

-e는 오류가 발생하면 스크립트를 종료하는 옵션이라 다음의 명령어를 실행하면 Hello까지만 출력되고 프로그램이 종료된다.

```
#!/bin/bash
set -e

echo "Hello"
aaa
echo "World"
```

```
[testuser@lms sh4]$ sh hello.sh
Hello
hello.sh: line 5: aaa: command not found
```

- 옵션 끄기 : 옵션을 꺼야 하는 경우에는 +를 이용할 수 있다.

```
#!/bin/bash

set -x

echo "AA"
ls -alh ./

# 위의 명령을 실행 할 때는 디버깅 옵션이 출력되지만 아래 명령을
# 실행할 때는 출력되지 않는다.

set +x

echo "BB"
ls -alh ./
```

```
[testuser@lms sh4]$ sh opx.sh
+ echo AA
AA
+ ls -alh ./
합계 16K
drwxrwxr-x.  2 testuser testuser   51  8월  25 10:50 .
drwx------. 13 testuser testuser 4.0K  8월  25 10:50 ..
-rw-rw-r--.  1 testuser testuser   50  8월  25 10:47 hello.sh
-rw-rw-r--.  1 testuser testuser  206  8월  25 10:50 opx.sh
-rw-rw-r--.  1 testuser testuser  162  8월  25 09:41 read.sh
+ set +x
BB
합계 16K
drwxrwxr-x.  2 testuser testuser   51  8월  25 10:50 .
drwx------. 13 testuser testuser 4.0K  8월  25 10:50 ..
-rw-rw-r--.  1 testuser testuser   50  8월  25 10:47 hello.sh
-rw-rw-r--.  1 testuser testuser  206  8월  25 10:50 opx.sh
-rw-rw-r--.  1 testuser testuser  162  8월  25 09:41 read.sh
```

9. 리스트 변수 (배열)

- 선언: 변수명=(데이터1 데이터2 데이터3)
- 사용: ${변수명[인덱스번호]}

  (인덱스번호는 0이 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 선언
echo ${filelist[*]} # $filelist 모든 데이터 출력
```

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

```
#!/bin/bash

id=("leeminsung" 20 "student")
echo ${id[*]} # $id의 모 든 데 이 터 출 력
```

```
leeminsung 20 student
```

10. 연산자

expr: 숫자 계산

- expr를 사용하는 경우 역 작은 따옴표(`)를 사용해야한다.
- 연산자 *와 괄호 ( )앞에는 역 슬래시를 같이 사용해야한다. (₩: 붙여 쓴다)
- 연산자와 숫자, 변수, 기호 사이에는 space를 넣어야 한다.

```
#!/bin/bash

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

echo $num
```

11. read

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 .
```

12. 문자열

- 문자열의 대문자화와 소문자화는 다음과 같이 처리한다. (Uppercase, Lowercase)

```
#!/bin/bash

str="abcd"
echo ${str^^}
echo ${str,,}
```

```
[testuser@lms sh4]$ sh uplo.sh
ABCD
abcd
```

- 문자열 변경 (replace)

${변수명/문자A/문자B}	첫번째 문자A를 문자B로 변경
${변수명//문자A/문자B}	모든 문자A를 문자B로 변경

```
[testuser@lms sh4]$ rep=a.b
[testuser@lms sh4]$ echo ${rep/./_}
a_b
[testuser@lms sh4]$ echo ${rep//./_}
a_b
```

13. 파이프, 리다이렉션: 표준 입력, 출력, 에러를 연결하기 위해서 사용한다.

- 파이프: command1 | command2 와 같은 형태로 사용되고, command1 의 표준 출력을 command2 의 표준 입력으로 전달한다. &를 붙이면 표준 에러도 함께 전달한다.

```
$ # read.sh의 내용을 읽어서, grep의 입력으로 전달
$ cat read.sh | grep a
```

- 리다이렉션: command > filename와 같은 형태로 사용되고, 파일을 읽어서 표준입력으로 전달하거나, 표준 출력을 파일로 저장한다. 파일 대신 명령의 결과를 입력, 출력할 수도 있다. &를 붙이면 표준 에러도 함께 전달한다.

>	기존에 있는 파일 내용을 지우고 저장
>>	기존 파일 내용 뒤에 덧붙여서 저장
<	파일의 데이터를 명령에 입력

- 2>&1 와 /dev/null

스크립트를 작성할 때 2>&1 이라는 표현을 볼 수 있다. 이 명령은 표준 에러(2)를 표준 출력(1)으로 리다이렉션 하는 것이다. 간단하게 말하면, 보통 프로그램에서 에러가 발생하면, 화면에 에러 메시지를 표시해서 사용자에게 경고를 주게 된다. 2>&1 은 에러가 발생하면 프로그램이 작동을 멈추거나 꺼지게 하지 않게 하고, 대신 에러내용을 표준 출력 동작으로 행동하게 하여 프로그램은 오류가 있더라도 화면에 경고메세지를 출력하지 말고 파일이나 딴 곳으로 처리하고, 계속 실행하라는 의미로 볼 수 있다.

리눅스는 표준 입력과 출력을 숫자로 표현할 수 있다. 이를 파일 디스크립터라고 한다.

표준 입력	0
표준 출력	1
표준 에러	2

/dev/null은 표준 출력을 버리기 위한 용도로 사용되는 디스크립터이다. 간단하게 말하면, /dev/null은 블랙홀이라고 이해할 수 있다. 이 경로에 보내지는 모든 파일과 데이터들은 없어져서 화면에 표시되지 않는다. 처리 결과로 출력되는 로그를 보지 않기 위해 사용하고, 2>&1 과 함께 사용하여 표준 출력, 표준 에러를 화면에 표시하지 않는 용도로 사용한다.

```
# sample.sh의 표준 출력을 보이지 않도록 리다이렉션
$ sample.sh > /dev/null
```

```
# sample.sh의 표준 출력, 에러를 보이지 않도록 리다이렉션
$ sample.sh > /dev/null 2>&1
```

> /dev/null 2>&1 은 스크립트의 출력 결과도 에러 내용도 /dev/null에 리다이렉션 시켜서 버린다는 뜻이다.

14. date 함수

- 문법

date +형식지정

date +"%형식지정"

date +"%형식지정%형식지정"

date +"%형식지정-%형식지정"

```
[testuser@lms sh4]$ date +"%m-%d-%y"
08-25-22
```

```
[testuser@lms sh4]$ date +"%H"
14
[testuser@lms sh4]$ date +"%H%M"
1457
[testuser@lms sh4]$ date +"%H:%M"
14:57
```

(오후 2 시 57 분)

- 특정 날짜 구하기: -d 옵션

  -d 구할 날짜의 문자열 (ex. date -d – 1 hours)

15. 조건문 (if)

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

```
if [ 조건 ]
then
    명령문
fi
```

- 두 인자값을 받아서 두 인자값이 다르면 different values 출력

```
#!/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 if조건시 [[]]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)

- 파일검사

-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

- 쉘 스크립트로 해당 파일이 있는지 없는지 확인

```
#!/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

- 두 인자 값을 받아서 두 인자 값이 같으면 same values, 다르면 different values가 출력

```
#!/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
```

- 조건문 한 줄에 작성하기

```
[lms@localhost dev]$ if [ 조건 ]; then 명령문 ; fi
[lms@localhost dev]$ if [ -z $1 ]; then echo 'Insert arguments'; fi
```

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

- ping : 서버에서 여러가지 컴퓨터가 연결되어 있을 때 연결된 특정 컴퓨터가 정상적으로 동작 하는지, 꺼져 있는지, 비정상적으로 동작하는지 확인하는 명령이다. 해당 컴퓨터의 ip주소로 ping명령어를 실행하고, 그 주소에 확인 요청을 한다. 정상적인 컴퓨터는 응답을 하지만 비정상적인 컴퓨터는 응답을 하지 않는다.

```
#!/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!
```

16. 반복문

- **for문**: for문은 주어진 배열에 데이터가 있는 동안 순차적으로 반복된다. 반복 중에 if문과 continue, break문을 이용하여 for문의 처음으로 돌아가거나, 탈출하는 것이 가능하다.

```
#!/bin/bash

for [ 배 열 _아 이 템  ] in [ 배 열  ]
do
        명 령 1
        ${배 열 _아 이 템 }
done
```

- 숫자 데이터를 이용한 반복: 연속된 숫자를 반복하는 방법은 다음과 같다. 1 에서 100 까지의 숫자를 반복한다.

```
#!/bin/bash

for vTime in {1..100}
do
        echo ${vTime}
done
```

- 포멧에 맞는 숫자 반복

  포멧에 맞게 숫자를 반복할 수도 있다. 00 에서 23 까지의 형식으로 숫자를 반복하는 방법은 다음과 같다.

```
#!/bin/bash

# 방 법  1
for vTime in {00..23}
do
        echo ${vTime}
done

# 방 법  2
for vTime in 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23
do
        echo $vTime
done
```

- 배열 데이터를 이용한 반복

```
#!/bin/bash

vArray=(A
B
C)

for vItem in "${vArray[@]}"
do
         echo $vItem
done
```

```
[testuser@lms sh4]$ sh arr.sh
A
B
C
```

```
#!/bin/bash

vArray=(A B C)

for vItem in "${vArray[@]}"
do
        echo $vItem
done
```

```
[testuser@lms sh4]$ sh arr2.sh
A
B
C
```

- 문자열을 분할하여 반복

  입력된 문자열을 분할하여 반복하는 방법도 있다. 배쉬셀에 문자열을 분리할 때 기준이 되는 IFS변수*를 이용하여 처리한다.

```
#!/bin/bash

vStrs="min 1 c
sung 2 b
lee 3 a"
IFS=$'\n' # 라 인 단 위 로  분 리
vArray=($vStrs)
IFS=$" " # 공 백 을  기 준 으 로  분 리

for vItem in "${vArray[@]}"
do
        echo "------"
        arr=($vItem)
        echo "name=${arr[0]}"
        echo "rank=${arr[1]}"
        echo "type=${arr[2]}"
done
```

```
[testuser@lms sh4]$ sh str.sh
------
name=min
rank=1
type=c
------
name=sung
rank=2
type=b
------
name=lee
rank=3
type=a
```

*IFS변수: IFS는 Internal Field Separator의 약자로 외부프로그램을 실행할 때 입력되는 문자열을 나눌 때 기준이 되는 문자를 정의하는 환경변수이다. 터미널에서 환경변수를 출력해보면 공백문자가 출력 됨을 볼 수 있다.

```
[testuser@lms sh4]$ echo $IFS

```

IFS의 디폴트 값은 공백/탭/개행문자 (space/tab/newline)이다.

쉘 스크립트에서 for in 문법을 보면, 공백문자로 띄워진 하나의 문자열이 마치 배열처럼 하나씩 순회하는 것을 볼 수 있다.

```
#!/usr/bin/bash

mystring="foo bar baz rab"

for word in $mystring; do
        echo "Word: $word"
done
```

```
[testuser@lms sh4]$ sh ifs.sh
Word: foo
Word: bar
Word: baz
Word: rab
```

IFS값이 공백이라 공백에 따라 단어가 쪼개진다는 것은 알았다. 그럼 IFS값을 바꾸면 어떻게 될까?

```
#!/usr/bin/bash

IFS=':' # 문자열 구분을 :로 한다.
mystring="foo bar baz rab"

for word in $mystring; do
        echo "Word: $word"
done
```

```
[testuser@lms sh4]$ sh ifs.sh
Word: foo bar baz rab
```

결과에서 볼 수 있듯이, 그대로 하나의 문자열이 출력되었다. 단어를 쪼개는 기준이 되는 문자가 공백에서 :로 바뀌었기 때문이다. 문자열을 "foo:bar:baz:rab"로 바꾸면 처음처럼 순회가 될 것이다.

- 현재 디렉토리에 있는 파일과 디렉토리 출력

```
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
```

- **while문:** while문의 기본 문법은 다음과 같다. 조건이 참일동안 명령 1 과 명령 2 가 순차적으로 반복된다. 명령을 처리하는 중간에 if문과 continue, break문을 이용하여 while문의 처음으로 돌아가거나, 탈출하는 것이 가능하다.+

```
#!/bin/bash

while [ 조건  ]
do
        명령 1
        명령 2
done


while [ 조건  ]; do 명령 1;명령 2; done
```

- 기본 루프 처리

  기본적인 루프문 처리는 다음과 같다. number가 2 보다 작거나 같을 동안(le) 반복된다.

```
#!/bin/bash

number=0

while [ $number -le 2 ]
do
        echo "Number: ${number}"
        ((number++))
done
```

```
[testuser@lms sh4]$ sh bl.sh
Number: 0
Number: 1
Number: 2
```

- 무한 루프

  무한 루프는 다음과 같이 while : 로 표현된다. if문을 이용하여 2 보다 커지면 while문을 탈출한다.

```
#!/bin/bash

number=0

while :
do
        if [ $number -gt 2 ]; then
                break
        fi

        echo "Number: ${number}"
        ((number++))
done
```

```
[testuser@lms sh4]$ sh infi.sh
Number: 0
Number: 1
Number: 2
```

- 날짜를 이용한 루프

  시작일자(2019.01.01)부터 종료일자 전일(2019.01.31)까지 일자를 출력하는 방법은 다음과 같다.

```
#!/bin/bash

startDate=`date +"%Y%m%d" -d "20190101"`
endDate=`date +"%Y%m%d" -d "20190201"`

while [ "$startDate" != "$endDate" ] ;
do
        echo $startDate

        startDate=`date +"%Y%m%d" -d "$startDate + 1 day"`;
done
```

```
[testuser@lms sh4]$ sh wl.sh
20190101
20190102
20190103
20190104
20190105
20190106
20190107
20190108
20190109
20190110
20190111
20190112
20190113
20190114
20190115
20190116
20190117
20190118
20190119
20190120
20190121
20190122
20190123
20190124
20190125
20190126
20190127
20190128
20190129
20190130
20190131
```

종료일자(2019.02.01)까지 출력하기 위해서는 종료일자에 1 을 더하여 while문 종료조건을 늘려주면 된다.

```
#!/bin/bash

startDate=`date +"%Y%m%d" -d "20190101"`
endDate=`date +"%Y%m%d" -d "20190201"`
endDate=`date +"%Y%m%d" -d "${endDate} + 1 day"`

while [ "$startDate" != "$endDate" ] ;
do
        echo $startDate
        startDate=`date +"%Y%m%d" -d "$startDate + 1 day"`;
done
```

```
[testuser@lms sh4]$ sh wl2.sh
20190101
20190102
20190103
20190104
20190105
20190106
20190107
20190108
20190109
20190110
20190111
20190112
20190113
20190114
20190115
20190116
20190117
20190118
20190119
20190120
20190121
20190122
20190123
20190124
20190125
20190126
20190127
20190128
20190129
20190130
20190131
20190201
```

17. for문과 while문의 차이

for문을 사용하는 경우	while문을 사용하는 경우
반복횟수가 정해진 경우 (주로 배열과 함께 많이 사용)	무한루프나 특정조건에 만족할 때까지 반복해야 하는 경우 (주로 파일을 읽고 쓰기에 많이 사용)

18. 환경설정

배쉬쉘의 설정 값을 설정하는 파일은 .bashrc와 .bash_profile 두 개가 있다. 사용자가 로그인할 때 .bash_profile 파일을 호출하고, 이 파일에서 .bashrc 파일을 호출한다. 사용자가 로그인 시 호출 순서는 .bash_profile → .bashrc → /etc/bashrc이다. 보통 .bashrc 파일에 사용자가 필요한 alias나 환경 변수를 기록한다.

- .bash_profile 파일의 기본내용이다. 배쉬쉘 환경인지 확인하여 .bashrc파일을 호출한다.

```
#!/bin/bash
if [ -n "$BASH" ] && [ -f ~/.bashrc ] && [ -r ~/.bashrc ]; then
        source ~/.bashrc
fi
```

- .bashrc 파일의 기본내용이다. 운영체제의 배쉬쉘 설정 파일을 확인하고 호출한다.

```
# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi
```

19. 프롬프트 스트링 (PS)

프롬프트 스트링은 쉘에서 사용자의 입력을 대기할 때 나타나는 문자이다. 4 가지 종류가 있다.

- PS1
  - 기본 프롬프트 스트링
  - 기본값은 [₩u@₩h₩W]₩$
- PS2
  - 긴 문자 입력을 위해 나타나는 문자열
  - 기본값은 >
- PS3
  - select 옵션을 처리할 때 나타나는 문자열
- PS4
  - 실행을 디버깅할 때 출력되는 문자열
  - 기본값은 +

가장 많이 보게 되는 것은 PS1 이며 이를 설정하는 방법에 대해 알아보겠다. PS1 은 사용자의 입력을 대기할 때 [scott@home var]$와 같은 형태로 나오는 프롬프트이다. PS1 번수를 전역 변수로 export하면 된다. 색상 설정을 함께 하여 명령어 입력과 구분하여 주는 것이 좋다. PS1 을 설정할 때 특수기호를 이용하여 이름을 다이나믹하게 변경할 수 있다.

₩u	사용자명
₩h	호스트명
₩W	현재 디렉토리명

```
[testuser@lms sh4]$ purple="\[\033[0;35m\]"
[testuser@lms sh4]$ white="\[\033[1;37m\]"
[testuser@lms sh4]$ green="\[\033[1;32m\]"
[testuser@lms sh4]$
[testuser@lms sh4]$ non_color="\[\033[0m\]"
[testuser@lms sh4]$
[testuser@lms sh4]$ export ps1="[$green\u$white@$purple\h$white \W]\$$non_color"
```

20. 참고사항

- AWS: 아마존 웹 서비스로 아마존의 클라우드 컴퓨팅 사업부이다.

- wc: 파일내의 단어 수 등의 정보를 출력한다. 아무런 옵션을 주지 않고서 사용하면 행수, 단어수, 문자수를 모두 검사해서 보고한다.
- wc -l: 행의 숫자를 알고 싶을 때 사용한다
- wc -c: 문자의 개수만을 알고 싶을 때 사용한다
- wc -w: 단어의 개수만을 알고 싶을 때 사용한다

```
[testuser@lms sh4]$ wc ifs.sh
  8  20 139 ifs.sh
```