일일 업무 사항 정리

작성자	제품팀 이민성 인턴
업무 일시	2022.08.25

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 쉘 스크립트 언어의 부족한 개념과 이미 알고 있던 개념 등을 공부해봤습니다. 파이프, 리다이렉션, 2>&1 와 /dev/null, env, set, export, PS, 환경설정, 특수 표현, 전달 피라미터, 문자열 대소문자화, while, for, while과 for의 차이점, if, date, ifs, aws, wc 등에 대해 공부했습니다. 하지만 프롬프트 스트링(PS)에 대해 이해가 잘 가지 않고 실제로 쓰는지 모르겠습니다. 그리고 while문의 특정 파일 생성을 대기와 for문의 여러 경로의 데이터를 조회하는 것이 잘 이해가 되지 않았습니다. 참고는 하려고 적어 놨으나 나중에 aws에 대해 자세히 공부를 하게 되면 다시 봐야 할 것 같습니다. 새롭게 알게 된 개념이 많아서 모르던 게 많았다는 것을 다시 한번 느꼈습니다. 앞으로 더 열심히 하겠습니다.	내일은 오늘 배운 것을 포함해 그동안 배웠던 쉘에 대한 것들을 다시 한번 정리해보고 상기시키겠습니다. 이미 배웠어도 잘 모르는 개념이 있기에 한번 더 상기시키면 좋을 것 같습니다. 그리고 반복문 (for과 while) 개념이 부족해 여러 예제를 통해 연습해봐야 할 것 같습니다. 차후에 공부를 더 열심히 한 뒤 미로 찾기 게임을 구현해보겠습니다.

2. 내용 세부 (업무 세부 내역 정리 및 기타 사항 정리)

1. 배쉬 쉘 빌트인 명령

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

- 파이프 : command1 | command2 와 같은 형태로 사용되고, command1 의 표준 출력을 commad2 의 표준 입력으로 전달한다. &를 붙이면 표준 에러도 함께 전달한다.

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

- 리다이렉션 : command > filename 와 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 표준에러(2)를 표준출력(1)으로 리다이렉션 하는 것이다. 간단하게 말하면, 보통 프로그램에서 에러가 발생하면, 화면에 에러 메세지를 표시해서 사용자에게 경고를 주게 된다. 2>&1 은 에러가 발생하면 프로그램이 작동을 멈추거나 꺼지지 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 버려버린다는 뜻이다.

2. 환경변수 설정 : env, set, export

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
```

환경 변수 설정의 현재의 세션에만 유효하다. 모든 세션에 적용하기 위해서는 .bashrc나 .profile 같은 설정 파일에 선언해야 한다.

- set 명령어 옵션

-a	생성, 변경되는 변수를 export 함
-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

다음의 명령어를 실행하면 Hello 까지만 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
```

3. 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 변수를 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"
```

4. 환경설정

배쉬쉘의 설정값을 설정하는 파일은 .bashrc와 .bash_profile 두 개가 있다. 사용자가 로그인할 때 .bash_profile 파일을 호출하고, 이 파일에서 .bashrc 파일을 호출한다.

사용자 로그인시 호출 순서는 .bash_profile > .bashrc > /etc/bashrc 이다. 보통 .bashrc 파일에 사용자가 필요한 alias나 환경 변수를 기록한다.

- .bash_profile

.bash_profile 파일의 기본내용이다. 배쉬쉘 환경인지 확인하여 .bashrc 파일을 호출한다.

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

- .bashrc

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

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

5. 특수 표현

- 특수 표현

$$	쉘의 PID 출력
$?	마지막으로 실행한 명령어의 실행 상태 (0: 정상, 0 이 아니면 오류 상태 출력)
$!	마지막으로 실행한 명령어의 PID출력

- 전달 파라미터

$0	쉘 파일의 이름
$1	첫번째 파라미터
$*	전체 파라미터
$#	파라미터의 개수

2. 쉘 스크립트 작성

1. 문자열

- Uppercase, Lowercase

문자열의 대문자화와 소문자화는 다음과 같이 처리합니다.

```
#!/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
```

2. while 루프

- 문법

while 문의 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
```

- 사용예제

- 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 표현한다. 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
```

- 특정 파일 생성을 대기

스크립트를 이용해서 작업할 때 특정 파일이 생성되는 것을 대기했다가 작업을 처리해야 하는 경우가 있다. 이럴때는 while문과 sleep명령을 이용해서 파일이 생기는 것을 대기했다가 작업을 처리하면 된다.

```
#!/bin/bash

vFileLocation="s3://bucket-name/sample.file"

# vFileLocation을 2분 에  한 번 씩  체 크 해 서  파 일 이  생 기 면  종 료
while :
do
        vCount=`aws s3 ls ${vFileLocation} | wc -l`
        if [[ "${vCount}" == "1" ]]; then
                echo "file check done."
                break
        fi

        echo "file waiting..."
        sleep 2m
done
```

3. for

- 문법

for문은 주어진 배열에 데이터가 있는 동안 순차적으로 반복된다. 반복중에 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
```

- 사용예제

- 숫자 데이터를 이용한 반복

  연속된 숫자를 반복하는 방법은 다음과 같다. 1 에서 100 까지의 숫자를 반복하는 예제이다.

  ```
  #!/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
  배열을 이용한 반복은 다음과 같다.

```
#!/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. 배쉬셸에 문자열을 분리할 때 기준이 되는 IFS변수를 이용하여 처리한다.

```
#!/bin/bash

vStrs="min 1 c
sung 2 b
lee 3 a"
IFS=$'\n' # 라인단위로 분리
vArray=($vStrs)
IFS=$" " # 공백을 기준으로 분리

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
```

- 여러 경로의 데이터를 조회할 때

스크립트를 이용하여 여러 경로의 데이터를 한 번에 조회하거나, 다른 처리를 해야하는 경우가 있다. 이럴때는 아래와 같이 사용한다. 다음은 vTargetdate, vTargethour를 입력받거나, 자동으로 설정되어 vLocations의 여러 경로의 데이터를 aws 커맨드로 조회할 때 사용하는 예제이다.

```
#!/bin/bash

# 사용자가 Targetdate, Targethour를 입력하지 않으면 자동으로 설정
if [ $# == 3 ]; then
        vTargetdate=$1
        vTargethour=$2
else
        vTargerdate=`date -d -1hour -u +\%Y\%m\%d`
        vTargethour=`date -d -1hour -u +\%H`
fi

echo "Targetdate: "${vTargetdate}
echo "Targerhour: "${vTargethour}

# 여러 경로를 한번에 확인할 때
vLocations=(/A/B/C/D/${vTargetdate}/${vTargetHour}/
/A/B/C/D/${vTargerdate}/${vTargethour}/
/A/B/C/D/${vTargetdate}/${vTargethout}/)

for vLocation in "${vLocations[@]}"
do
        aws s3 ls s3://bucket-name/${vLocations}
done
```

4. if문 테스트 명령

배쉬셀에서 기본적으로 제공하는 파일 확인 명령에 대해 알아보겠다. 다음의 표현은 if문에서 사용할 수 있다.

-e	파일이 존재하면 true
-f	일반 파일이면 true
-s	파일이 0 사이즈가 아니면 true
-d	디렉토리면 true
-b	블록 디바이스면 true
-c	캐릭터 디바이스면 true
-L	심볼링 링크면 true
-r	읽기 권한이 있으면 true
-w	쓰기 권한이 있으면 true
-x	실행 권한이 있으면 true
-z	문자열 길이가 0 이면 true

- 사용예제

```
# 파일이 심볼릭 링크이면 A 출력
[ -L /path/file ] && echo "A"

# 파일이 없으면 B출력
[ ! -f /path/file ] && echo "B"


# 문자열 길이가 0이면 출력
[ -z "" ] && echo "A"
```

5. 모르는 개념들 정리

- **date 함수**
  - 문법

    date +형식지정

    date +"%형식지정"

    date +"%형식지정%형식지정"

    date +"%형식지정-%형식지정"

  - 예제

```
[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

- **IFS**

  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

```

  IFS는 디폴트 값은 공백/탭/개행 문자다. (space, tab, new line)
  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.

- 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
```

6. for과 while의 차이점

- for문을 사용하는 경우 : 반복횟수가 정해진 경우, 주로 배열과 함께 많이 사용
- while문을 사용하는 경우 : 무한루프나 특정 조건에 만족할 때까지 반복해야 하는 경우, 주로 파일을 읽고 쓰기에 많이 사용