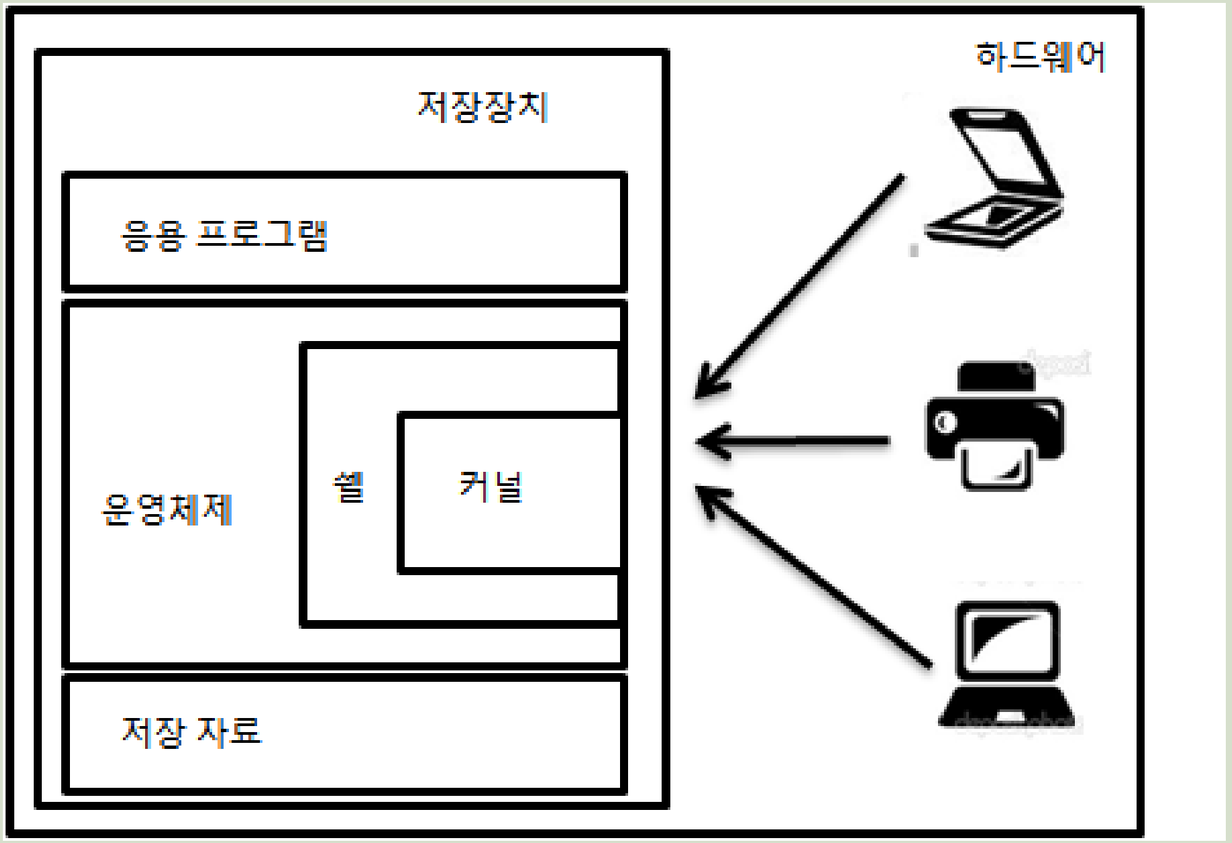
아바라 23-03-13

Bash 쉘 스크립트

Bash 쉘 스크립트 문법

유닉스 쉘

쉘이란? 명령어와 프로그램을 실행 할때 사용하는 인터페이스
sh, ksh, csh, bash등 이 있다.



변수 선언

```
#!/usr/bin/bash

name="dongbok" # 변수 선언 및 대입
pass=123123 # 따옴표로 감싸든 말든 문자열로 저장됨

echo $name # {}가 있으나 없으나 $만으로 변수의 값을 넣어줄 수 있으나, 문자열을 붙여서 쓸려면 ${} 를 사용해야 한다.
echo "my name is mr.${name}"
printf "%s" $pass
```

지역변수 & 전역변수

```shell
# 기본적으로 전역 변수로 지정
string="hello world"

function string_test() {
    # local을 붙여야 지역변수로 인식. 만일 local을 빼면 전역변수 덮어쓰기가 되버림
    local string="hello local @@"
    echo ${string}
}

# 함수 호출
string_test # > hello local @@
echo ${string} # > hello world

# 변수 초기화
unset string
```

타입 지정

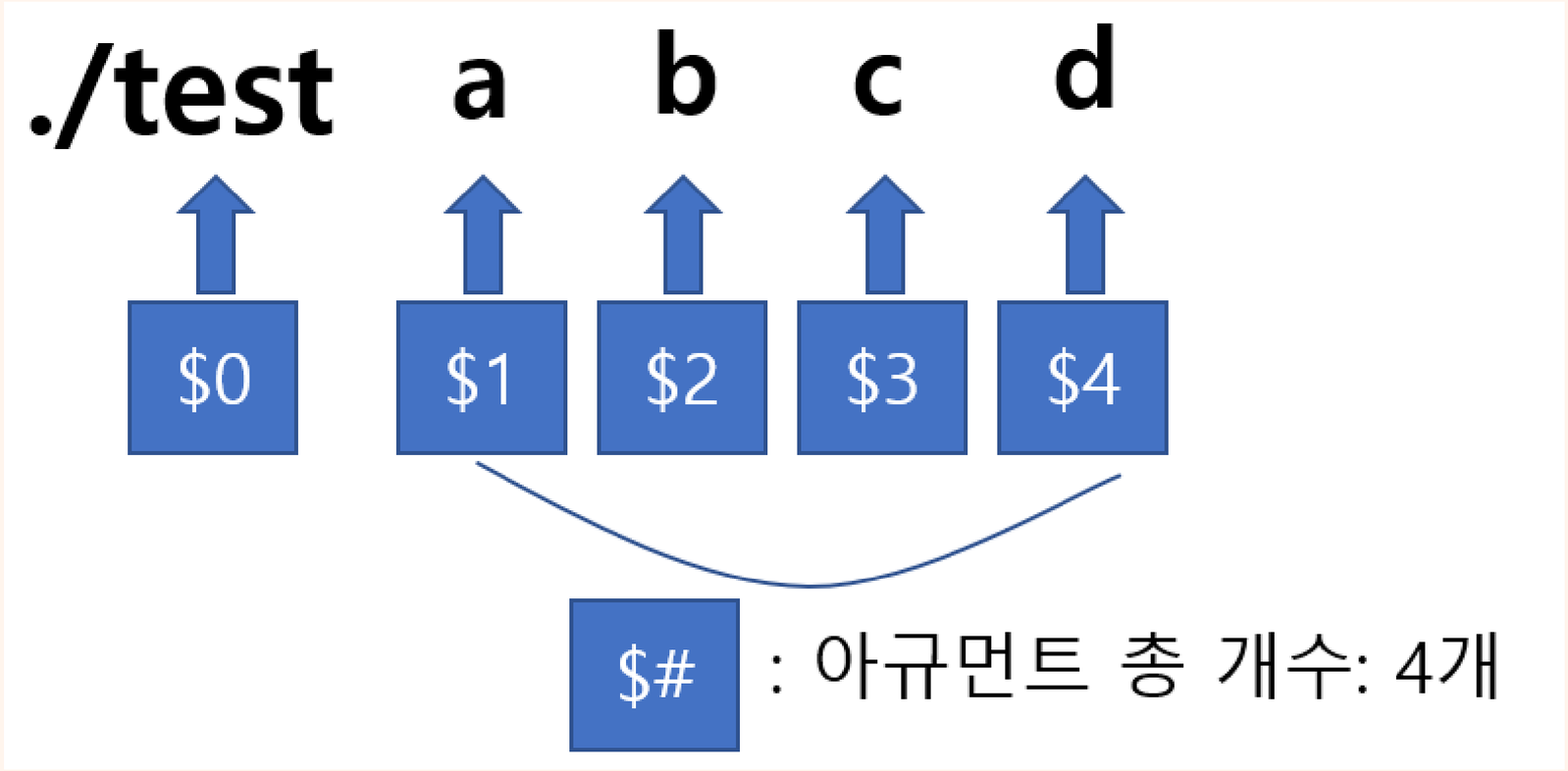
```
# -r 읽기 전용 타입 (상수 const라고 보면 된다)
declare -r var1
readonly var1

# -i 정수형 타입
declare -i number
number=3
echo "number = $number"     # number = 3

# -a 배열 타입
declare -a indices

# -A 연관배열(MAP) 타입
declare -A map

# -f 함수 타입
declare -f

# -x 환경변수(export) 지정
declare -x var3 # 스크립트 외부 환경에서도 이 변수를 쓸 수 있게 해준다.
```

환경 변수

```
# /home/export_test.sh 파일을 만들고 작성

#!/usr/bin/bash

echo ${hello_world}
```

```
# 환경 변수 선언
export hello_world="global hello world"

# 자식 스크립트 호출은 스크립트 경로을 쓰면된다.
/home/export_test.sh
# > 자식스크립트의 코드에서 부모스크립트에서 정의한 hello_world변수값이 출력된다.
```

매개 변수



매개 변수

종류	설명
$0	실행된 셸 스크립트명
$1	스크립트에 넘겨진 첫 번째 아규먼트
$2	스크립트에 넘겨진 두 번째 아규먼트
$3 S4 등등...쭈욱 이후 $숫자	그 이후 해당되는 아규먼트
$#	아규먼트 개수
$*	스크립트에 전달된 인자 전체를 하나의 변수에 저장하면 IFS 변수의 첫 번째 문자로 구분
$@	$*와 동일한데 다른 점은 IFS 환경 변수를 사용하지 않는다는 점.
$!	실행을 위해 백그라운드로 보내진 마지막 프로그램 프로세스 번호
$$	셸 스크립트의 PID
$?	실행한 뒤의 반환 값 (백그라운드로 실행된 것 제외)

쉘 산술 연산

Bash에서는 계산을 처리할 수 있는 expr, let, $(()) 3가지 문법을 제공한다.

```
#!/bin/bash

number1=10
number2=20

plus=`expr $number1 + $number2`
minus=`expr $number1 - $number2`
mul=`expr $number1 \* $number2` # 곱셈에는 \* 를 이용한다.
div=`expr $number1 / $number2`
rem=`expr $number1 % $number2`
```

조건문

```
if [ 값1 조건식 값2 ]
then
    수행1
else
    수행2
fi


# 가독성 좋기 위해 then을 if [] 와 붙여쓰려면 반드시 세미콜론 ; 을 써야한다.
if [ 값1 조건식 값2 ]; then
    수행1
else
    수행2
fi
```

비교연산

```
문자1 = 문자2          # 문자1 과 문자2가 일치 (sql같이 = 하나만 써도 일치로 인식)
문자1 == 문자2         # 문자1 과 문자2가 일치
문자1 != 문자2         # 문자1 과 문자2가 일치하지 않음
-z 문자                 # 문자가 null 이면 참
-n 문자                 # 문자가 null 이 아니면 참
문자 == 패턴           # 문자열이 패턴과 일치
문자 != 패턴           # 문자열이 패턴과 일치하지 않음
```

```
값1 -eq 값2            # 값이 같음(equal)
값1 -ne 값2            # 값이 같지 않음(not equal)
값1 -lt 값2            # 값1이 값2보다 작음(less than)
값1 -le 값2            # 값1이 값2보다 작거나 같음(less or equal)
값1 -gt 값2            # 값1이 값2보다 큼(greater than)
값1 -ge 값2            # 값1이 값2보다 크거나 같음(greater or equal)
```

논리연산

```
조건1 -a 조건2          # AND
조건1 -o 조건2          # OR
조건1 && 조건2          # 양쪽 다 성립
조건1 || 조건2          # 한쪽 또는 양쪽다 성립
!조건                    # 조건이 성립하지 않음
true                     # 조건이 언제나 성립
false                    # 조건이 언제나 성립하지 않음
```

if elif else문

```bash
#!/bin/bash

num1="10"
num2="10"

if [ ${num1} -lt ${num2} ]; then # "-lt", A가 B보다 작으면 True
    echo "yes"
elif [ ${num1} -eq ${num2} ]; then # "-eq", A와 B가 서로 같으면 True
    echo "bbb"
else
    echo "no"
fi
```

case문

```
case 문자열 in
 경우1)
    명령 명령 명령
    ;;
 경우2)
    명령 명령 명령
    ;;
 * )
    명령 명령 명령
    ;;
esac
```

```
case ${var} in
    "linux") echo "리눅스" ;; # 변수var값이 linux라면 실행
    "unix") echo "유닉스" ;;
    "windows") echo "윈도우즈" ;;
    "MacOS") echo "맥OS" ;;
    *) echo "머야" ;; # default 부분
esac
```

반복문

```bash
#!/bin/bash

# 초기값; 조건값; 증가값을 사용한 정통적인 for문
for ((i=1; i<=4; i++)); do
    echo $i
done
```

반복문

```bash
#!/bin/bash

# 루프 돌 데이터에 띄어쓰기가 있으면 각각 돌음
for x in 1 2 3 4 5
do
    echo "${x}"
done


# 변수를 사용한 반복문
data="1 2 3 4 5"
for x in $data
do
    echo ${x}
done


# 배열을 사용한 반복문
arr=(1 2 3 4 5)
for i in "${arr[@]}" # arr[@] : 배열 전체 출력
do
    echo "${i}"
done
```

```bash
#!/bin/bash

# sequence를 통한 for문. seq라는 프로세스가 순서대로 숫자를 출력해 주는 역할을 bash에 사용한 것이다.
for num in `seq 1 5`
do
  echo $num
done


# range를 사용한 반복문. {..} 중괄호와 점 두개를 쓰면 range처리가 된다.
for x in {1..5}
do
    echo ${x}
done
```

Q & A