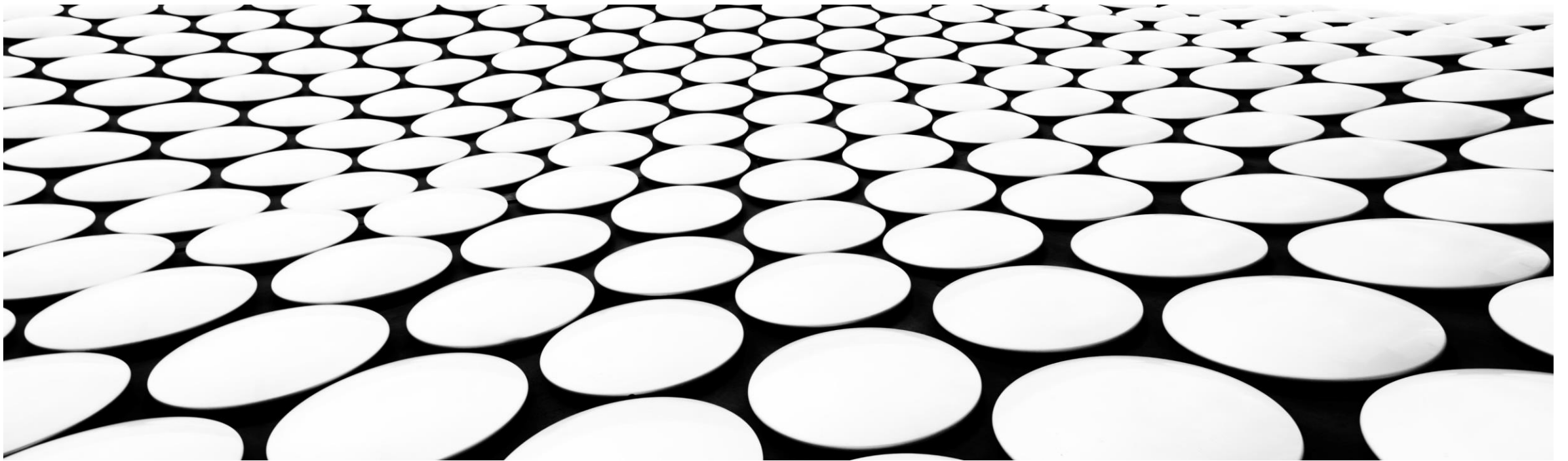

LINUX의 셸 스크립트 및 MAKEFILE



LINUX - SHELL SCRIPT

특수 권한

- 일반적으로 리눅스 유닉스는 사용자의 파일 권한을 부여하여 기초적인 보안체계를 유지한다. 파일이나 디렉터리에서는 user, group, other 권한이 존재하며 각각 읽기, 쓰기, 실행 권한을 부여할 수 있다.
- 이러한 권한을 수정하기 위해서 chmod 명령을 사용하는데 셸에서 특수 권한을 주기 위해서도 같은 명령을 사용한다.

LINUX - SHELL SCRIPT

SetUID

- 파일을 실행할 때 일시적으로 소유자의 권한을 얻어 실행할 수 있도록 한다.
- root 권한으로 지정된 프로그램에 SetUID가 지정되어 있으면 root 권한으로 실행된다.
- 기존에 실행 권한이 없으면 대문자 S, 있으면 소문자 s로 표시된다.

```
$ touch setuid
$ ll
-rw-r--r--  1 seongwon  staff    0B  1 25 14:50 setuid

$ chmod 4644 ./setuid
-rwsr--r--  1 seongwon  staff    0B  1 25 14:50 setuid
```

LINUX - SHELL SCRIPT

SetUID

- 리눅스에서 설정되어 있는 대표적인 파일은 /usr/bin/passwd이다.
 - 해당 파일은 계정의 비밀번호를 변경할 수 있도록 하는 실행파일로 root만 변경 가능하도록 설정되어 있다.
 - /usr/bin/passwd에 SetUID가 설정되어 있지 않으면 일반 사용자는 root 사용자를 통해 비밀번호를 변경해야 하므로 일반 사용자로 /etc/passwd 파일을 수정 가능하도록 설정되어 있다.
 - /etc/passwd는 사용자의 이름과 같은 권한에 대한 데이터 베이스만 존재하며, 실제 비밀번호는 /usr/bin/passwd 사용자가 패스워드를 변경할 때만 사용한다. 과거 패스워드가 파일 형태로 저장되어 있으나 현재는 저장되지 않는다. 이러한 부분에 대한 자세한 설명은 shadow파일의 대해 참조하길 바란다.

```
$ ls -l /usr/bin/passwd  
-rwsr-xr-x 1 root root 68208 May 28 2020 /usr/bin/passwd
```

LINUX - SHELL SCRIPT

SetGID

- 파일을 실행할 때 일시적으로 파일 소유 그룹의 권한을 얻어 실행할 수 있도록 한다.
 - 기존 권한에 실행 권한이 없으면 대문자 S, 있으면 소문자 s로 표시된다.

```
$ touch setgid
$ ll
-rw-r--r-- 1 seongwon staff 0B 1 25 15:20 setgid

$ chmod 2644 ./setuid
-rw-r-Sr-- 1 seongwon staff 0B 1 25 15:20 setgid
```

LINUX - SHELL SCRIPT

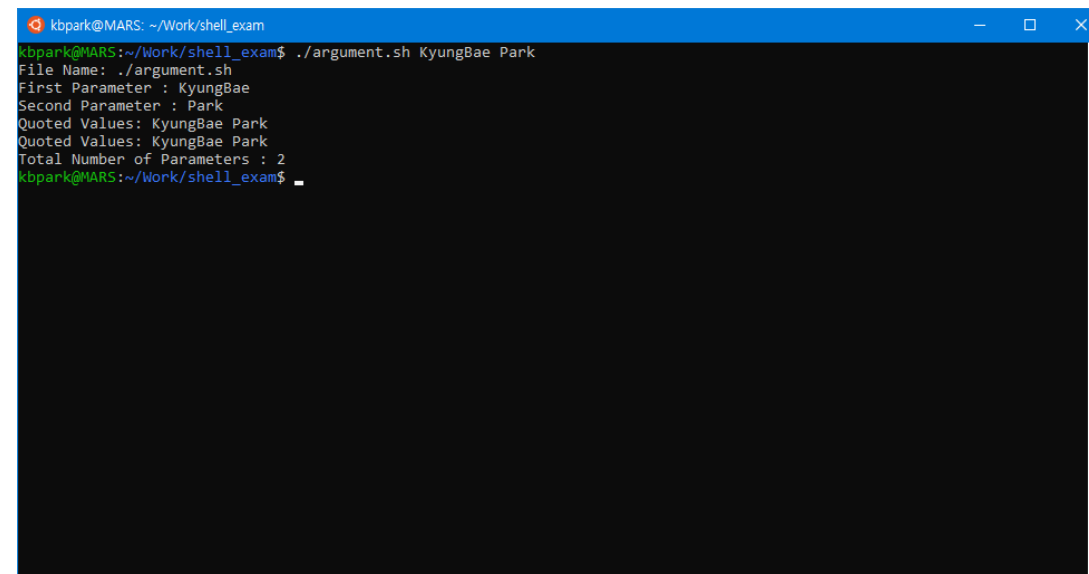
특수 변수

\$0	현재 스크립트 파일 이름
\$n	스크립트가 호출된 인수, n은 인수의 십진수를 표현 (\$1, \$2, \$3 ...)
\$#	매개 변수의 총 개수
\$*	전체 인자 값
\$@	모든 인자가 ""로 묶여 있으며, 스크립트가 두 개의 인수를 받으면 \$1, \$2와 동일하다.
\$?	마지막으로 실행된 명령어, 함수, 스크립트 자식의 종료 상태
\$\$	현재 스크립트의 PID
\$_	마지막으로 실행된 백그라운드 PID

LINUX - SHELL SCRIPT

Command-Line Arguments

- 위에서 설명한 특수 변수를 활용하여 쉘 스크립트를 작성하고, 인수를 넘긴다.
 - `#!/bin/bash`
 - `echo "File Name: $0"`
 - `echo "First Parameter : $1"`
 - `echo "Second Parameter : $2"`
 - `echo "Quoted Values: $@"`
 - `echo "Quoted Values: $*"`
 - `echo "Total Number of Parameters : $#"`

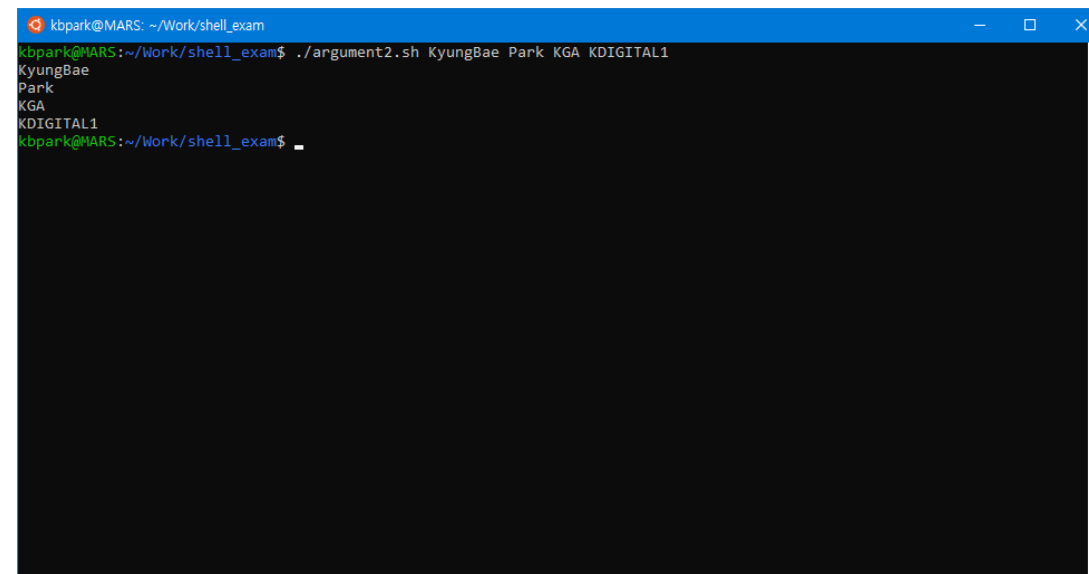


```
kbpark@MARS: ~/Work/shell_exam
kbpark@MARS:~/Work/shell_exam$ ./argument.sh KyungBae Park
File Name: ./argument.sh
First Parameter : KyungBae
Second Parameter : Park
Quoted Values: KyungBae Park
Quoted Values: KyungBae Park
Total Number of Parameters : 2
kbpark@MARS:~/Work/shell_exam$
```

LINUX - SHELL SCRIPT

Command-Line Arguments

- `#!/bin/bash`
- `for TOKEN in $*`
- `do`
- `echo $TOKEN`
- `done`



```
kbpark@MARS: ~/Work/shell_exam
kbpark@MARS:~/Work/shell_exam$ ./argument2.sh KyungBae Park KGA KDIGITAL1
KyungBae
Park
KGA
KDIGITAL1
kbpark@MARS:~/Work/shell_exam$
```


과제 - C++

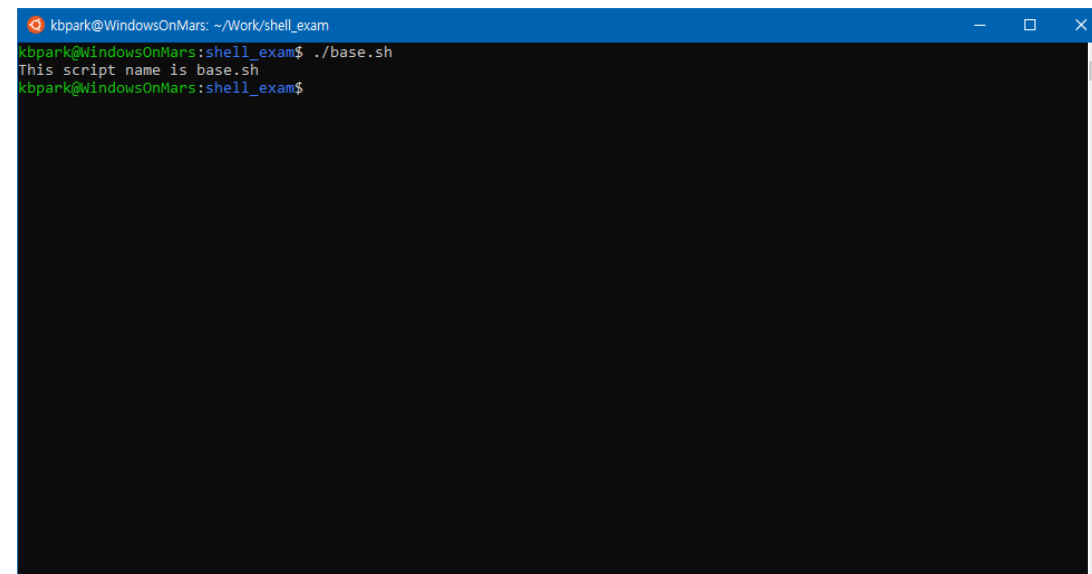
삼각형을 나타내는 객체를 만들기 위한 클래스를 선언하려고 한다. 삼각형은 세 개의 꼭짓점 좌표 (x[0], y[0]), (x[1], y[1]), (x[2], y[2])로 정의된다. 삼각형 객체는 다음과 같은 행동을 할 수 있다.

- 세 개의 꼭짓점 좌표(6개의 double형 값)를 매개 변수로 받아 삼각형 객체를 생성한다(생성자).
- 삼각형을 x축으로 dx, y축으로 dy만큼 이동할 수 있다.
(예) t.move(dx, dy); 와 같이 호출하면 삼각형 객체 t의 모든 꼭짓점 좌표들을 모두 (dx, dy)만큼 이동함
- 삼각형을 x축 방향으로 sx배, y축 방향으로 sy배 크기조정을 할 수 있다.
(예) t.scale(sx, sy); 와 같이 호출하면 삼각형 객체 t의 모든 꼭짓점의 x좌표를 sx배, y좌표를 sy배 크기조정한다.
- 삼각형의 면적을 구할 수 있다.
(참고) 삼각형의 면적은 다음과 같이 구할 수 있다.
면적 = $|(x[0] \times y[1] - y[0] \times x[1]) + (x[1] \times y[2] - y[1] \times x[2]) + (x[2] \times y[0] - y[2] \times x[0])| / 2$

LINUX - SHELL SCRIPT

명령어 치환(Command Substitution)

- 명령어 치환은 하나나 그 이상의 명령어의 출력을 재할당 해준다. 명령어 치환은 말그대로 한 명령어의 출력을 다른 문맥으로 연결해 준다.
- 명령어 치환의 두가지 형태
 - 역따옴표(`...`)를 쓰는 것이다
 - `$(...)` 양식
 - `#!/bin/bash`
- `script_name=`basename $0``
- `echo "This script name is $script_name"`



```
kbpark@WindowsOnMars: ~/Work/shell_exam
kbpark@WindowsOnMars:shell_exam$ ./base.sh
This script name is base.sh
kbpark@WindowsOnMars:shell_exam$
```

LINUX - SHELL SCRIPT

명령어 치환(Command Substitution)

- 명령어 치환은 Bash 에서 쓸 수 있는 툴셋을 확장시켜 줄 수 있다.
- 표준출력으로 결과를 출력해 주는(잘 동작하는 유닉스 툴이 그래야 하는 것처럼) 프로그램이나 스크립트를 짜고 그 결과를 변수로 할당하면 된다.

```
#include <stdio.h>
```

```
using namespace std;
```

```
/* "Hello, world." C++ program */
```

```
int main()
```

```
{
```

```
    cout << "Hello, world." endl;
```

```
    return (0);
```

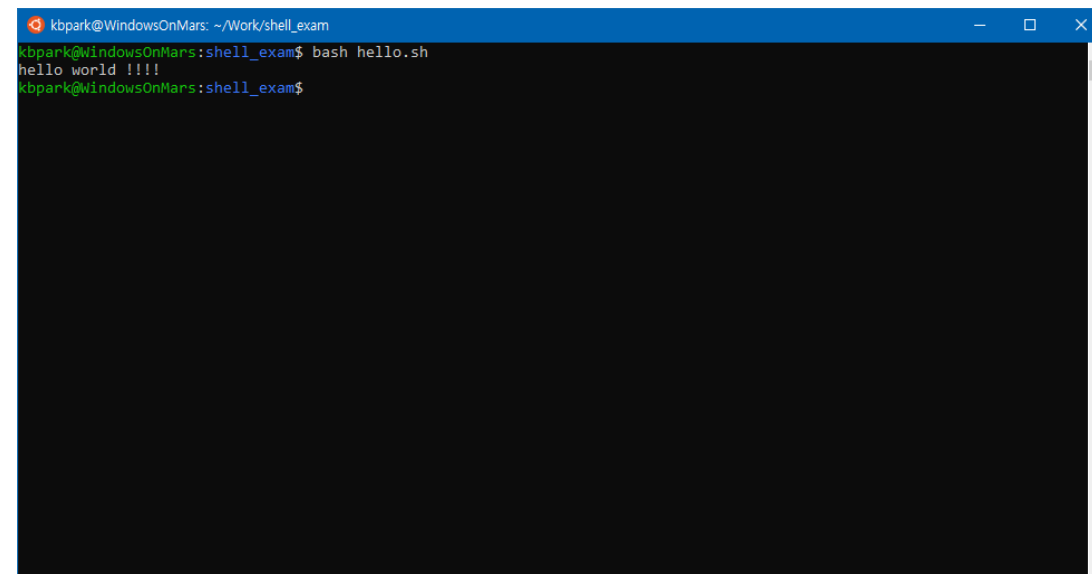
```
}
```

```
$ g++ -o hello hello.cpp
```

LINUX - SHELL SCRIPT

명령어 치환(Command Substitution)

- `#!/bin/bash`
- `# hello.sh`
- `greeting=`./hello``
- `echo $greeting`

A terminal window titled 'kbpark@WindowsOnMars: ~/Work/shell_exam' with a blue header bar. The terminal shows a shell prompt 'kbpark@WindowsOnMars:shell_exam\$' followed by the command 'bash hello.sh'. The output 'hello world !!!!' is displayed on the next line. The prompt then changes to 'kbpark@WindowsOnMars:shell_exam\$' again, indicating the script has finished execution. The terminal background is black with green text.

```
kbpark@WindowsOnMars: ~/Work/shell_exam
kbpark@WindowsOnMars:shell_exam$ bash hello.sh
hello world !!!!
kbpark@WindowsOnMars:shell_exam$
```

LINUX - SHELL SCRIPT

입력과 출력 리다이렉트

- 때로는 모니터에 표시된 명령의 출력을 저장할 필요가 있다 bash 셸은 명령의 출력을 다른 위치에(like a file) 리다이렉트할 수 있도록 몇 가지 연산자를 제공한다.
- 리다이렉트는 출력은 물론 입력에도 사용될 수 있으며 파일을 명령에 입력시킬수있다.

LINUX - SHELL SCRIPT

출력 리다이렉트

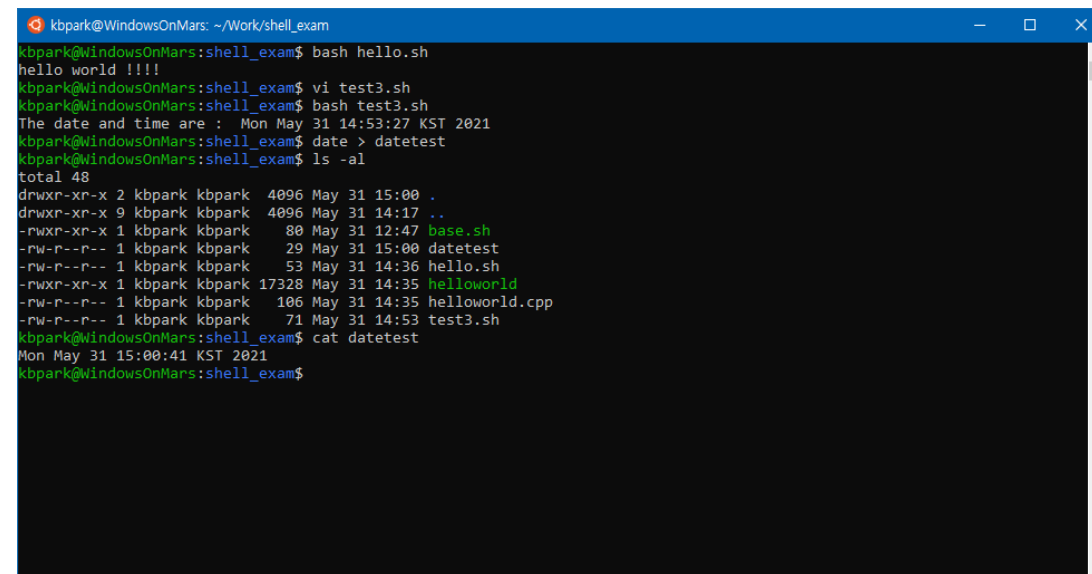
- 리다이렉트의 가장 기본적인 형태는 파일 명령의 출력을 전송한다.
- Bash 셸은 이를 위해 > 부등호 기호를 상용한다.
 - `$ command > outfile`
- 출력 파일에 저장되는 모니터에 표시할 수 있는 모든 명령의 출력은 지정된 파일에 대신 저장 된다.
 - `date > datetest`

LINUX - SHELL SCRIPT

출력 리다이렉트

- 출력 파일에 저장되는 모니터에 표시할 수 있는 모든 명령의 출력은 지정된 파일에 대신 저장 된다.

- `date > datetest`

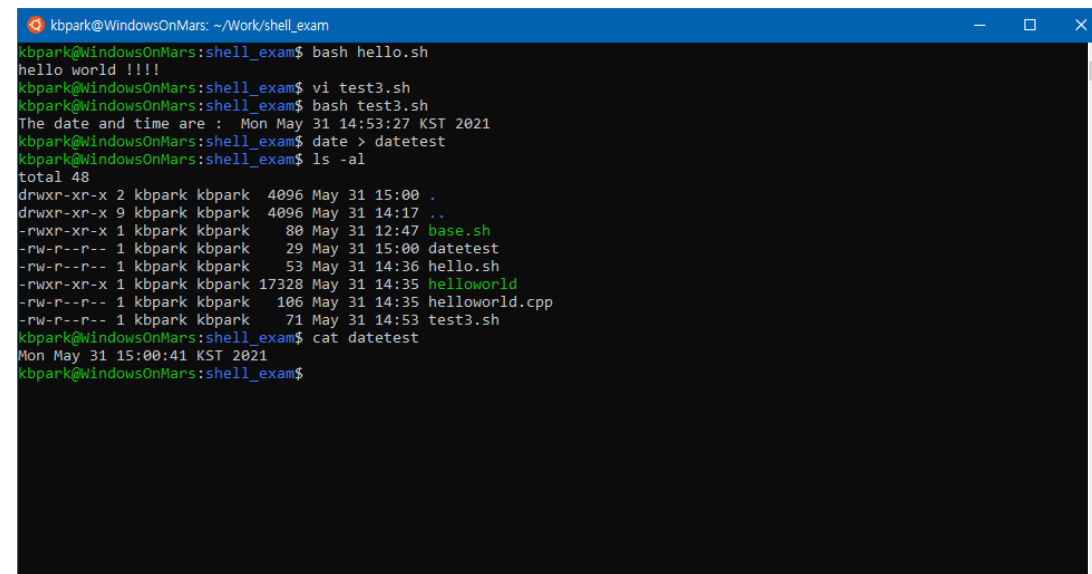


```
kbpark@WindowsOnMars: ~/Work/shell_exam
kbpark@WindowsOnMars:shell_exam$ bash hello.sh
hello world !!!!
kbpark@WindowsOnMars:shell_exam$ vi test3.sh
kbpark@WindowsOnMars:shell_exam$ bash test3.sh
The date and time are : Mon May 31 14:53:27 KST 2021
kbpark@WindowsOnMars:shell_exam$ date > datetest
kbpark@WindowsOnMars:shell_exam$ ls -al
total 48
drwxr-xr-x 2 kbpark kbpark 4096 May 31 15:00 .
drwxr-xr-x 9 kbpark kbpark 4096 May 31 14:17 ..
-rwxr-xr-x 1 kbpark kbpark 80 May 31 12:47 base.sh
-rw-r--r-- 1 kbpark kbpark 29 May 31 15:00 datetest
-rw-r--r-- 1 kbpark kbpark 53 May 31 14:36 hello.sh
-rwxr-xr-x 1 kbpark kbpark 17328 May 31 14:35 helloworld
-rw-r--r-- 1 kbpark kbpark 106 May 31 14:35 helloworld.cpp
-rw-r--r-- 1 kbpark kbpark 71 May 31 14:53 test3.sh
kbpark@WindowsOnMars:shell_exam$ cat datetest
Mon May 31 15:00:41 KST 2021
kbpark@WindowsOnMars:shell_exam$
```

LINUX - SHELL SCRIPT

출력 리다이렉트

- > 리다이렉트 연산자는 파일 datetest를 만들고 date 명령의 출력을 저장 했다.
- 출력 파일이 이미 존재하면 리다이렉트는 > 연산자는 기존의 파일을 새로운 파일데이터로 덮어 씩운다.
 - \$ whoami > datetest



```
kbpark@WindowsOnMars: ~/Work/shell_exam
kbpark@WindowsOnMars:shell_exam$ bash hello.sh
hello world !!!!
kbpark@WindowsOnMars:shell_exam$ vi test3.sh
kbpark@WindowsOnMars:shell_exam$ bash test3.sh
The date and time are : Mon May 31 14:53:27 KST 2021
kbpark@WindowsOnMars:shell_exam$ date > datetest
kbpark@WindowsOnMars:shell_exam$ ls -al
total 48
drwxr-xr-x 2 kbpark kbpark 4096 May 31 15:00 .
drwxr-xr-x 9 kbpark kbpark 4096 May 31 14:17 ..
-rwxr-xr-x 1 kbpark kbpark 80 May 31 12:47 base.sh
-rw-r--r-- 1 kbpark kbpark 29 May 31 15:00 datetest
-rw-r--r-- 1 kbpark kbpark 53 May 31 14:36 hello.sh
-rwxr-xr-x 1 kbpark kbpark 17328 May 31 14:35 helloworld
-rw-r--r-- 1 kbpark kbpark 106 May 31 14:35 helloworld.cpp
-rw-r--r-- 1 kbpark kbpark 71 May 31 14:53 test3.sh
kbpark@WindowsOnMars:shell_exam$ cat datetest
Mon May 31 15:00:41 KST 2021
kbpark@WindowsOnMars:shell_exam$
```


LINUX - SHELL SCRIPT

입력 리다이렉트

- 입력 리다이렉트는 출력과는 반대다.
- 입력 리다이렉트 파일은 파일의 내용을 받아서 명령으로 보낸다.
- 입력 리다이렉트 기호는 < 부등호 이다.
 - `$ command < infile`
- 기억하기 쉬운 방법은 언제나 커맨드라인의 처음에 나온다는 것이고, 리다이렉트 기호는 부등호가 아니라 데이터의 흐르는 방향을 '가리키는' 기호다.
- < 기호는 데이터가 입력 파일로부터 명령으로 흐르는 것을 나타낸다.
 - `$ wc < testinput`
- Wc 명령은 데이터의 텍스트 양을 계산한다. 기본적으로 3가지 값을 나타낸다.
 - 텍스트의 줄 수
 - 텍스트의 단어 수
 - 텍스트의 바이트 수

LINUX - SHELL SCRIPT

배열

- 우리가 지금까지 쉘 스크립트에서 변수를 선언할 때 다음과 같은 방법을 사용했다.
 - `#!/bin/bash`
 - `NAME01="Lucas"`
 - `NAME02="KyungBae"`
 - `NAME03="KGA"`
 - `NAME04="KDIGITAL"`
 - `echo $NAME01`
- 옆에서 정의한 변수를 배열로 정의하면 다음과 같이 사용할 수 있다.
 - `#!/bin/bash`
 - `NAME[0]="Lucas"`
 - `NAME[1]=" KyungBae"`
 - `NAME[2]=" KGA"`
 - `NAME[3]=" KDIGITAL"`
 - `echo "First Index: ${NAME[0]}"`
 - `echo "Second Index: ${NAME[1]}"`

LINUX - SHELL SCRIPT - 배열

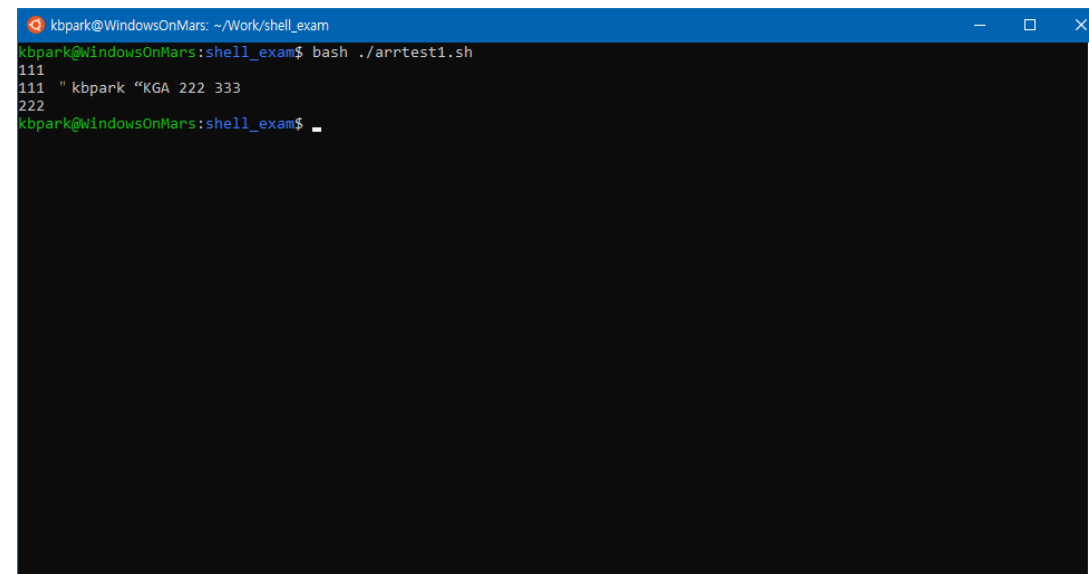
배열

- 만약 모든 배열에 접근하고자 한다면 다음 방법으로 수행할 수 있다.
 - `${array_name[*]}`
 - `${array_name[@]}`
- 셸 스크립트에서 배열을 정의하는 방법은 다음과 같다.
 - `array_name=("element 1" "element 2" "element 3")`

LINUX - SHELL SCRIPT - 배열

배열

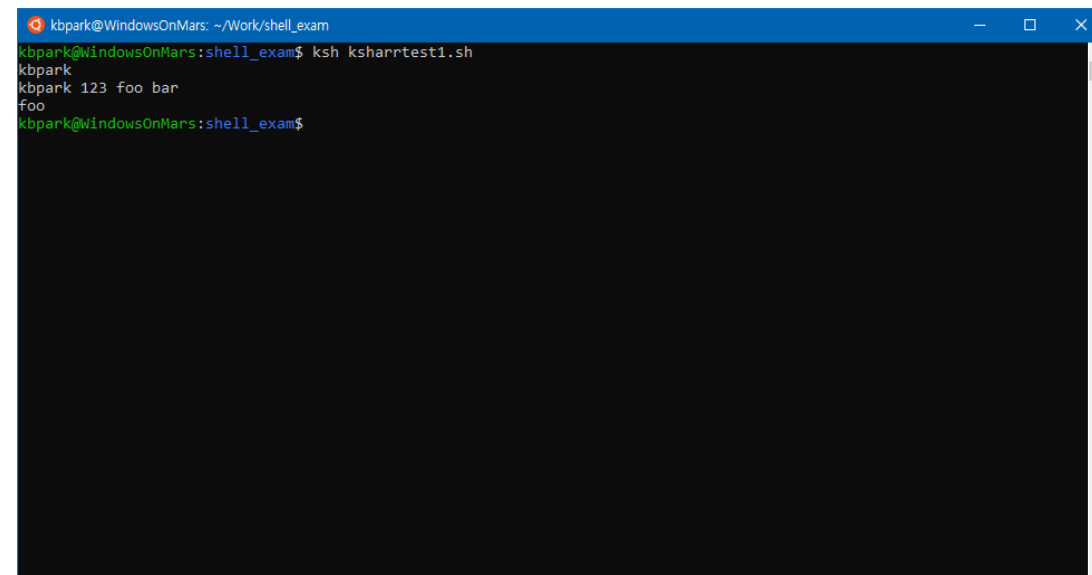
- 다음 예제를 수행하여 결과를 확인한다.
 - `#!/bin/bash`
 - `array=(111 " kbpark" "KGA" 222 333)`
 - `echo $array`
 - `echo ${array[*]}`
 - `echo ${array[2]}`
- 인덱스를 지정하지 않은 경우 첫 번째 값만 출력되며, 배열에 저장되어 있는 값을 모두 출력하거나 지정하여 출력할 수 있다.

A terminal window titled 'kbpark@WindowsOnMars: ~/Work/shell_exam' showing the execution of a shell script. The prompt is 'kbpark@WindowsOnMars:shell_exam\$'. The command entered is 'bash ./arrtest1.sh'. The output is: '111', '111 " kbpark "KGA 222 333', and '222'. The prompt returns to 'kbpark@WindowsOnMars:shell_exam\$'.

LINUX - SHELL SCRIPT - 배열

ksh 배열 정의

- ksh의 경우 bash와 다르게 set 명령을 사용하여 배열을 지정한다.
- 다음 예제를 확인한다.
 - `#!/bin/ksh`
 - `set -A array "lucas" 123 "foo" "bar"`
 - `echo $array`
 - `echo ${array[*]}`
 - `echo ${array[2]}`

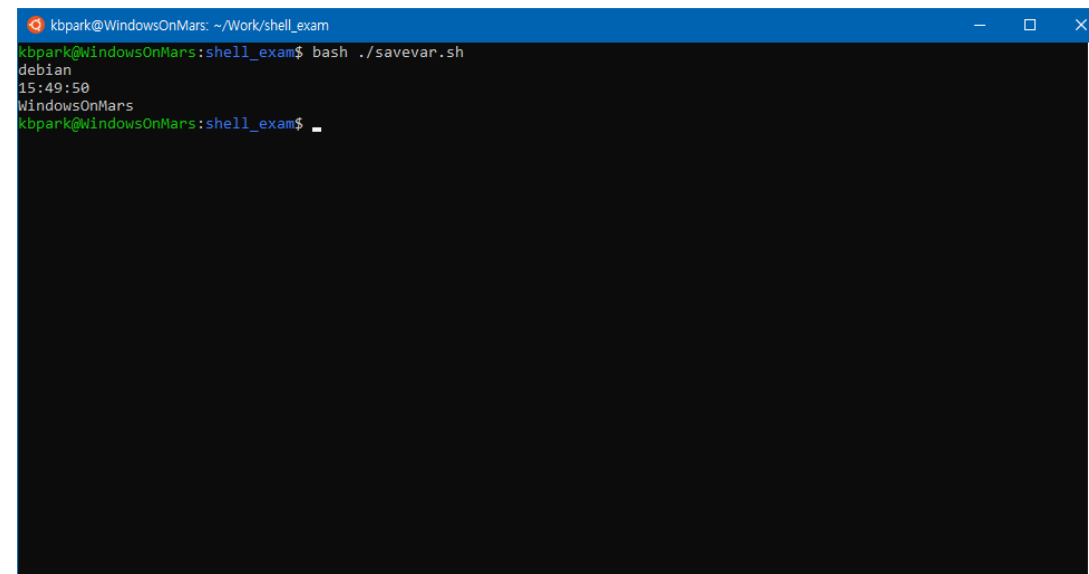
A terminal window titled 'kbpark@WindowsOnMars: ~/Work/shell_exam' with a blue header bar. The terminal shows the execution of a ksh script 'ksharrtest1.sh'. The prompt is 'kbpark@WindowsOnMars:shell_exam\$'. The script output is: 'kbpark', 'kbpark 123 foo bar', and 'foo'. The prompt returns to 'kbpark@WindowsOnMars:shell_exam\$' after the script finishes.

```
kbpark@WindowsOnMars: ~/Work/shell_exam
kbpark@WindowsOnMars:shell_exam$ ksh ksharrtest1.sh
kbpark
kbpark 123 foo bar
foo
kbpark@WindowsOnMars:shell_exam$
```

LINUX - SHELL SCRIPT - 배열

변수 값 혹은 명령 실행 결과 변수 저장하기

- 괄호안에 있는 값을 직접 지정할 수 있지만 변수로 설정한 값을 배열에 저장하거나, 역따옴표(`)를 통해서 명령의 결과를 저장할 수 있다.
 - `#!/bin/bash`
 - `VAR="redhat debian gentoo darwin"`
 - `DISTRO=($VAR)`
 - `echo ${DISTRO[1]}`
 - `TODAY=$(date)`
 - `echo ${TODAY[3]}`
 - `INFO=$(uname -a)`
 - `echo ${INFO[1]}`

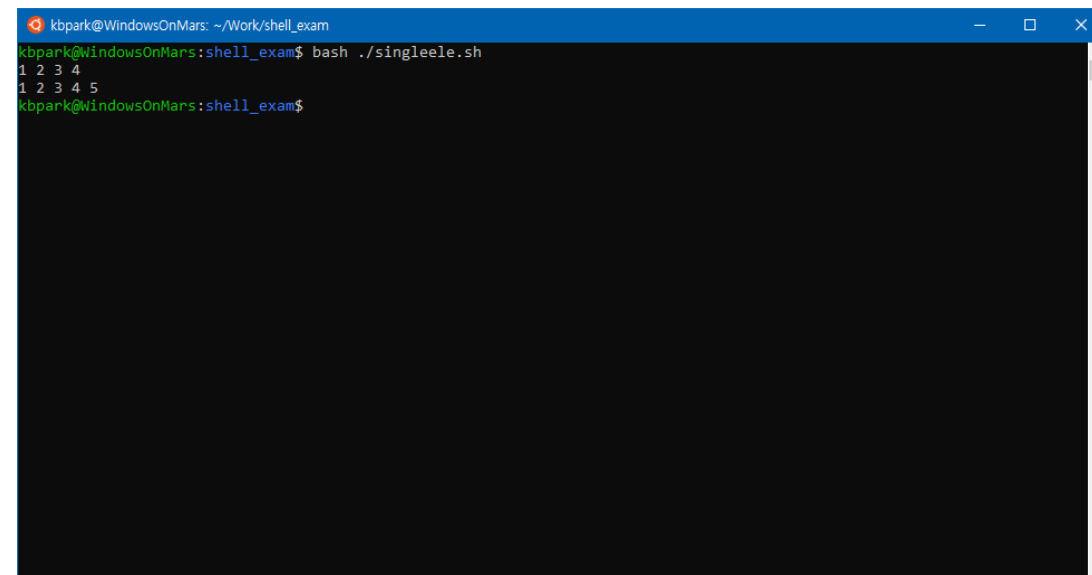


```
kbpark@WindowsOnMars: ~/Work/shell_exam
kbpark@WindowsOnMars:shell_exam$ bash ./savevar.sh
debian
15:49:50
WindowsOnMars
kbpark@WindowsOnMars:shell_exam$
```

LINUX - SHELL SCRIPT - 배열

배열 값 추가

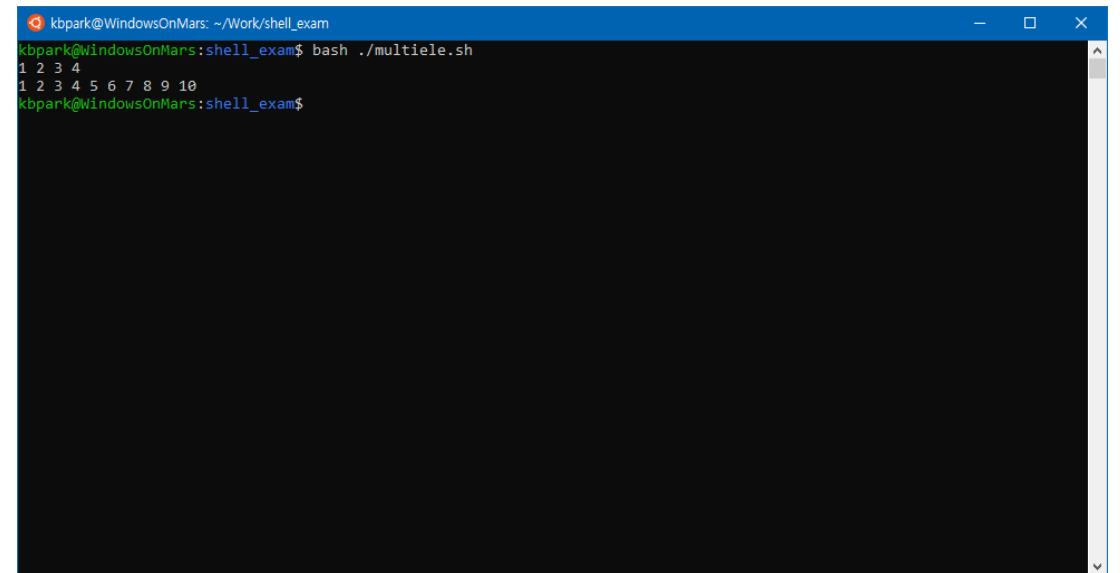
- 먼저 단일 요소를 추가하는 방법을 알아본다.
 - `#!/bin/bash`
 - `NUMBER=(1 2 3 4)`
 - `echo ${NUMBER[*]}`
 - `NUMBER+=(5)`
 - `echo ${NUMBER[*]}`

A terminal window titled 'kbpark@WindowsOnMars: ~/Work/shell_exam' showing the execution of a shell script. The prompt is 'kbpark@WindowsOnMars:shell_exam\$'. The command 'bash ./singleleele.sh' is entered. The output shows '1 2 3 4' on the first line and '1 2 3 4 5' on the second line. The prompt returns to 'kbpark@WindowsOnMars:shell_exam\$'.

LINUX - SHELL SCRIPT - 배열

배열 값 추가

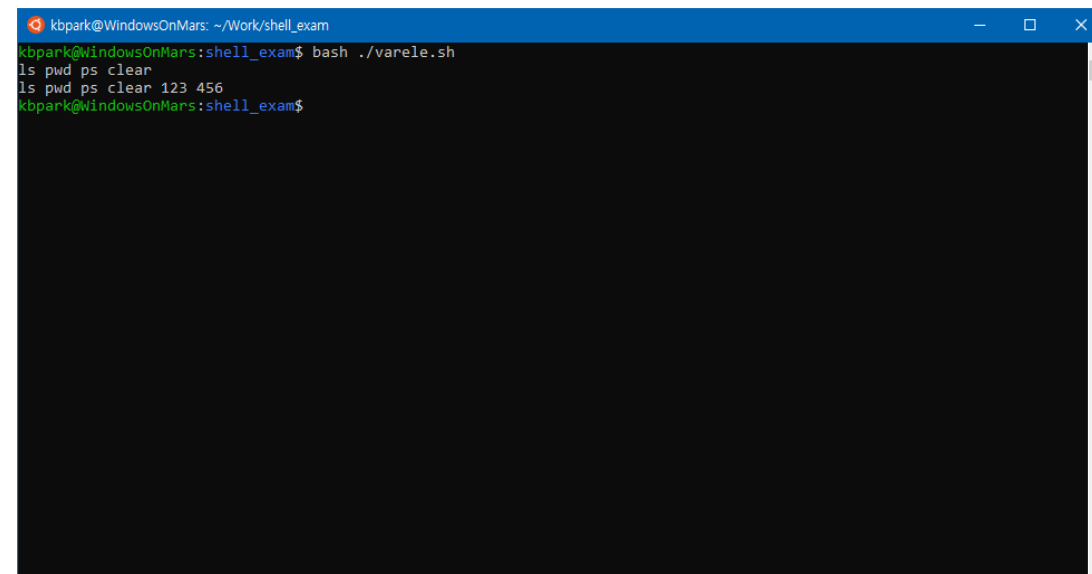
- 여러 요소를 추가하고 싶다면 다음과 같이 수행한다.
 - `#!/bin/bash`
 - `NUMBER=(1 2 3 4)`
 - `echo ${NUMBER[*]}`
 - `NUMBER+=(5 6 7 8 9 10)`
 - `echo ${NUMBER[*]}`

A terminal window titled 'kbpark@WindowsOnMars: ~/Work/shell_exam' showing the execution of a shell script. The prompt is 'kbpark@WindowsOnMars:shell_exam\$'. The command entered is 'bash ./multiele.sh'. The output shows two lines: '1 2 3 4' and '1 2 3 4 5 6 7 8 9 10'. The prompt returns to 'kbpark@WindowsOnMars:shell_exam\$'.

LINUX - SHELL SCRIPT - 배열

배열 값 추가

- 변수의 값을 요소로 추가하고자 하면 다음과 같이 수행한다.
 - `#!/bin/bash`
 - `COMMAND=("ls" "pwd" "ps" "clear")`
 - `echo ${COMMAND[*]}`
 - `ELEMENT="123 456"`
 - `COMMAND+=($ELEMENT)`
 - `echo ${COMMAND[*]}`

A terminal window titled 'kbpark@WindowsOnMars: ~/Work/shell_exam' with a blue header bar. The terminal shows the execution of a script named 'varele.sh'. The prompt is 'kbpark@WindowsOnMars:shell_exam\$'. The script content is displayed as follows:

```
kbpark@WindowsOnMars:shell_exam$ bash ./varele.sh
ls pwd ps clear
ls pwd ps clear 123 456
kbpark@WindowsOnMars:shell_exam$
```

LINUX - SHELL SCRIPT

기본 연산자



```
#!/bin/bash

greet() {
    echo "Hello, ${1}"
}

read -p "What is your name: " name
greet "$name"
```

LINUX - SHELL SCRIPT

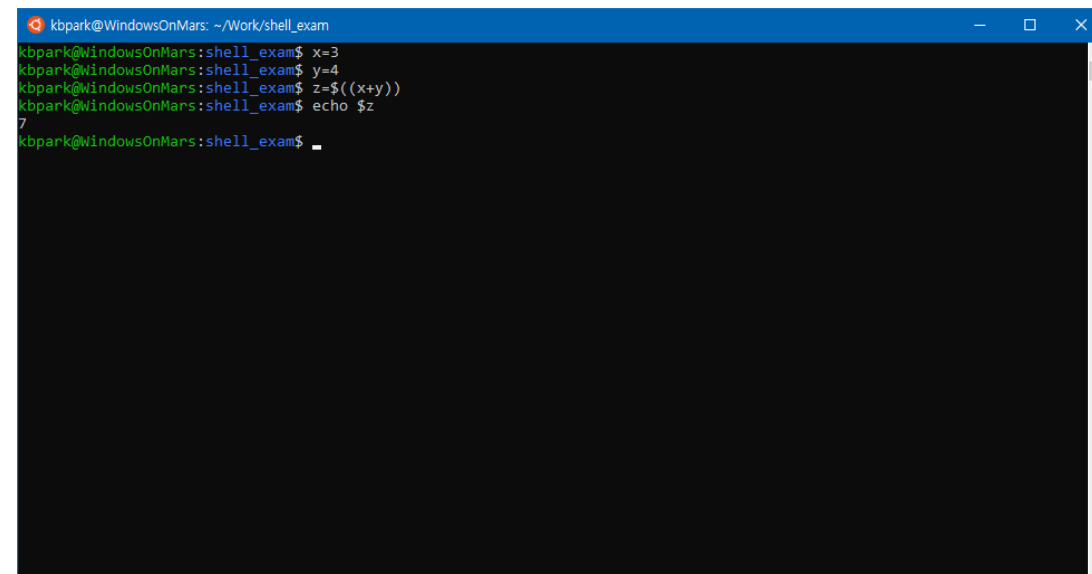
기본 연산자

- 다양한 셸 종류에 따라 연산자도 다양하지만 이번 시간에서는 가장 기본적인 bash 셸의 기본 연산자에 대해 설명한다.
 - 산술 연산자
 - 관계 연산자
 - Boolean 연산자
 - 문자열 연산자
 - 파일 테스트 연산자
- bash의 경우 간단한 산술 연산을 수행하는 메커니즘이 존재하지 않기 때문에 awk 혹은 expr과 같은 기본 명령을 사용한다.

LINUX - SHELL SCRIPT

bash 변수 처리

- `$ x=3`
- `$ y=4`
- `$ z=$((x+y))`
- `$ echo $z`
- `7`

A terminal window titled 'kbpark@WindowsOnMars: ~/Work/shell_exam' with a blue header bar. The window shows a series of shell commands and their outputs. The commands are: 'x=3', 'y=4', 'z=\$((x+y))', and 'echo \$z'. The output for the last command is '7'. The prompt is 'kbpark@WindowsOnMars:shell_exam\$' and the cursor is at the end of the line.

```
kbpark@WindowsOnMars: ~/Work/shell_exam
kbpark@WindowsOnMars:shell_exam$ x=3
kbpark@WindowsOnMars:shell_exam$ y=4
kbpark@WindowsOnMars:shell_exam$ z=$((x+y))
kbpark@WindowsOnMars:shell_exam$ echo $z
7
kbpark@WindowsOnMars:shell_exam$
```

LINUX - SHELL SCRIPT

bash 변수 처리

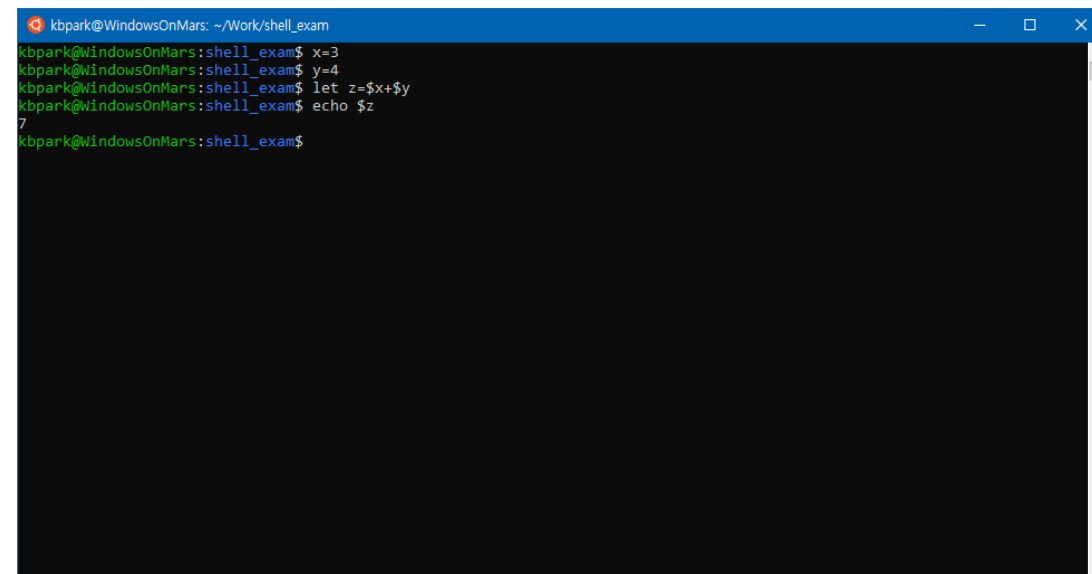
- `$ x=3`
- `$ y=4`
- `$ z=$((x+y))`
- `$ echo $z`
- `7`
- `echo $((x-y))`
- `echo $((x*y))`
- `echo $((x/y))`
- `# -1`
- `# 12`
- `# 0`
- `echo $((12/5))`
- `# 2`

```
kbpark@WindowsOnMars: ~/Work/shell_exam
kbpark@WindowsOnMars:shell_exam$ x=3
kbpark@WindowsOnMars:shell_exam$ y=4
kbpark@WindowsOnMars:shell_exam$ z=$((x+y))
kbpark@WindowsOnMars:shell_exam$ echo $z
7
kbpark@WindowsOnMars:shell_exam$ echo $((x-y))
-1
kbpark@WindowsOnMars:shell_exam$ echo $((x*y))
12
kbpark@WindowsOnMars:shell_exam$ echo $((x/y))
0
kbpark@WindowsOnMars:shell_exam$ echo $((12/5))
2
kbpark@WindowsOnMars:shell_exam$
```

LINUX - SHELL SCRIPT

let 명령어

- \$ x=3
- \$ y=4
- \$ let z=\$x+\$y
- \$ echo \$z
- 7

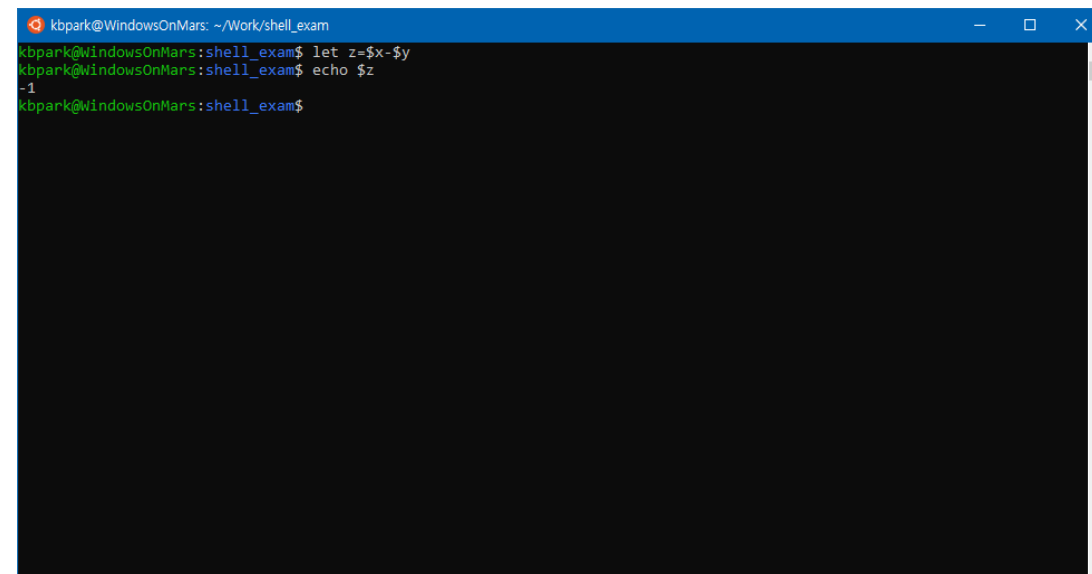
A terminal window titled 'kbpark@WindowsOnMars: ~/Work/shell_exam' with a blue header bar. The terminal shows a sequence of commands and their output: 'x=3', 'y=4', 'let z=\$x+\$y', and 'echo \$z'. The output of the last command is '7'.

```
kbpark@WindowsOnMars: ~/Work/shell_exam
kbpark@WindowsOnMars:shell_exam$ x=3
kbpark@WindowsOnMars:shell_exam$ y=4
kbpark@WindowsOnMars:shell_exam$ let z=$x+$y
kbpark@WindowsOnMars:shell_exam$ echo $z
7
kbpark@WindowsOnMars:shell_exam$
```

LINUX - SHELL SCRIPT

let 명령어

- `let z=$x-$y`
- `echo $z`
- `-1`

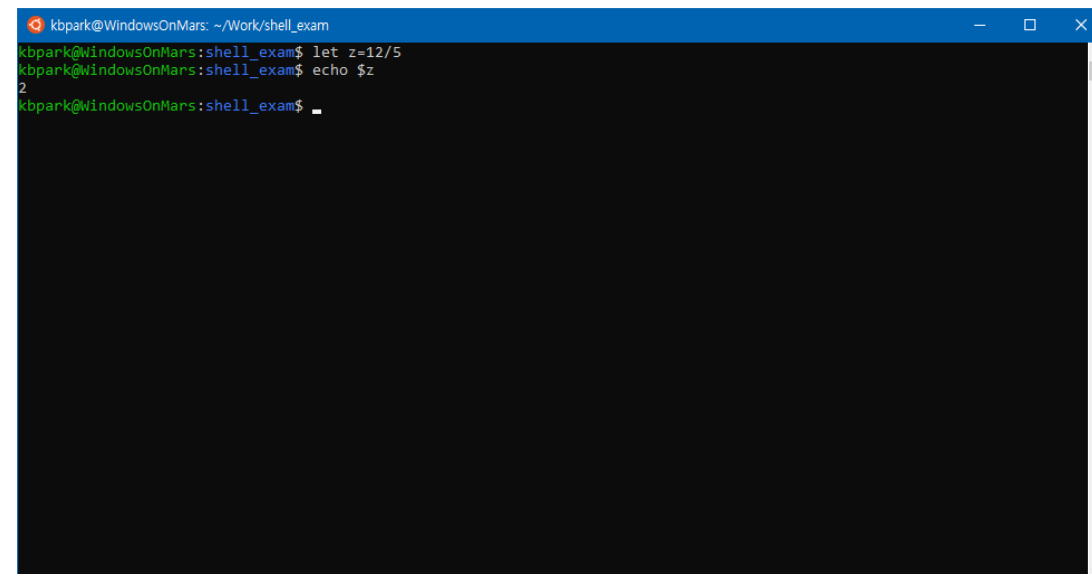
A terminal window titled 'kbpark@WindowsOnMars: ~/Work/shell_exam' with a blue header bar. The terminal shows the execution of a shell script. The first command is 'let z=\$x-\$y', followed by 'echo \$z'. The output of the echo command is '-1'. The prompt 'kbpark@WindowsOnMars:shell_exam\$' is visible at the end of each line.

```
kbpark@WindowsOnMars: ~/Work/shell_exam
kbpark@WindowsOnMars:shell_exam$ let z=$x-$y
kbpark@WindowsOnMars:shell_exam$ echo $z
-1
kbpark@WindowsOnMars:shell_exam$
```

LINUX - SHELL SCRIPT

let 명령어

- let z=12/5
- echo \$z
- # 2

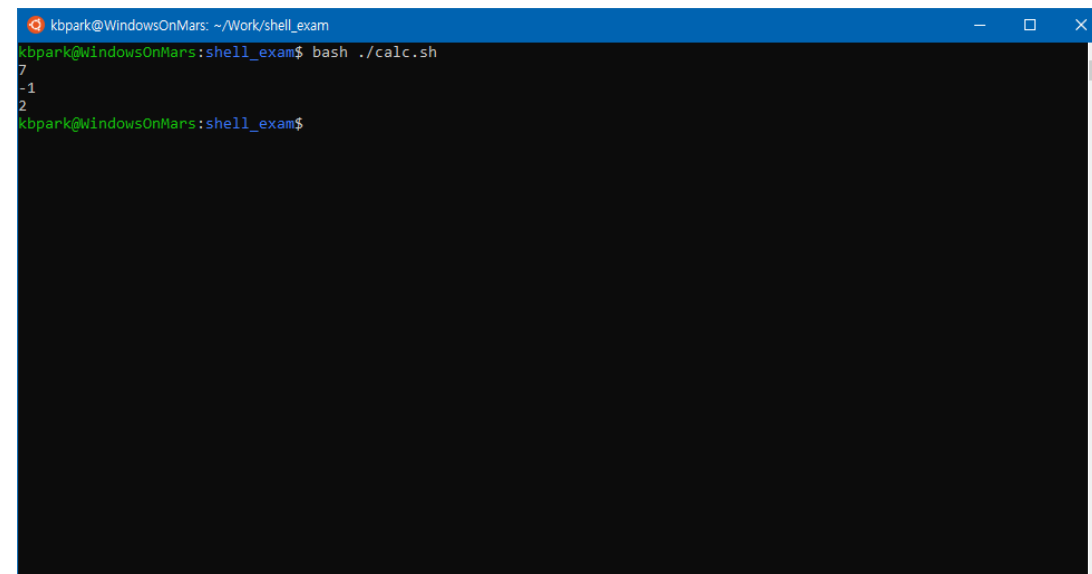
A terminal window titled 'kbpark@WindowsOnMars: ~/Work/shell_exam' with a blue header bar. The terminal shows the execution of a shell script. The first command is 'let z=12/5', followed by 'echo \$z'. The output of the echo command is '2'. The prompt returns to the shell.

```
kbpark@WindowsOnMars: ~/Work/shell_exam
kbpark@WindowsOnMars:shell_exam$ let z=12/5
kbpark@WindowsOnMars:shell_exam$ echo $z
2
kbpark@WindowsOnMars:shell_exam$
```


LINUX - SHELL SCRIPT

let 명령어

- `#!/bin/bash`
- `x=3`
- `y=4`
- `let z=$x+$y`
- `echo $z`
- `let z=$x-$y`
- `echo $z`
- `let z=12/5`
- `echo $z`

A terminal window titled 'kbpark@WindowsOnMars: ~/Work/shell_exam' with a blue header bar. The terminal shows a shell script being executed with the command 'bash ./calc.sh'. The script's output is displayed in green text: '7', '-1', and '2'. The prompt 'kbpark@WindowsOnMars:shell_exam\$' is visible at the bottom of the terminal.

```
kbpark@WindowsOnMars: ~/Work/shell_exam
kbpark@WindowsOnMars:shell_exam$ bash ./calc.sh
7
-1
2
kbpark@WindowsOnMars:shell_exam$
```

LINUX - SHELL SCRIPT

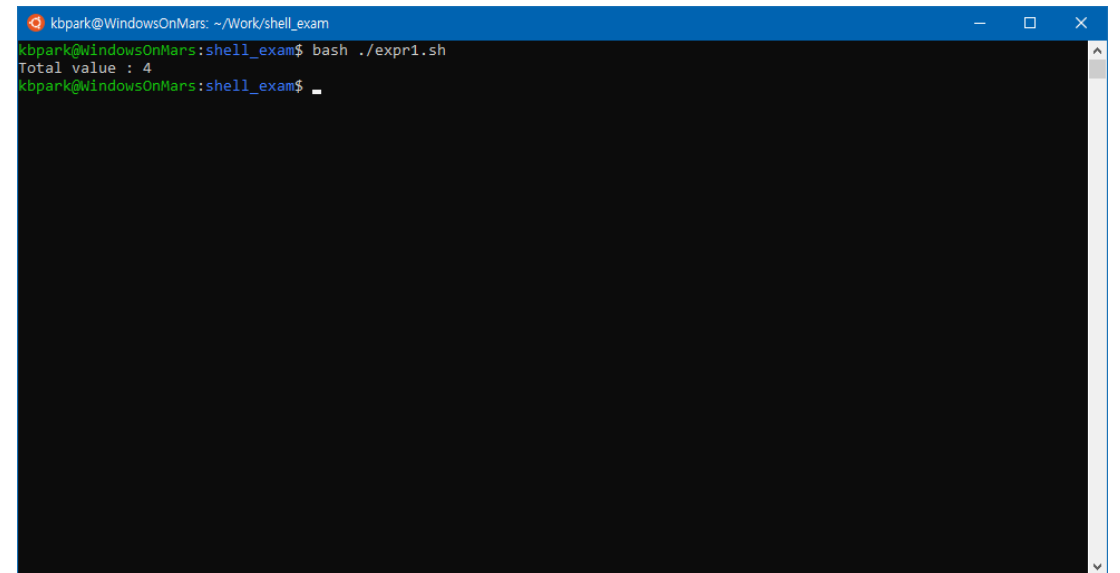
expr 명령어

- 숫자와 연산자 사이를 반드시 띄어 써야 한다.
- 예를 들어 2+2 형식으로 하면 동작하지 않으며 2 + 2 형식으로 작성해야 한다.
- 또한 쉘 스크립트 내에서 연산을 적용하기 위해서는 `` backtick을 추가해야 한다.

LINUX - SHELL SCRIPT

expr 명령어

- `#!/bin/sh`
- `val=`expr 2 + 2``
- `echo "Total value : $val"`

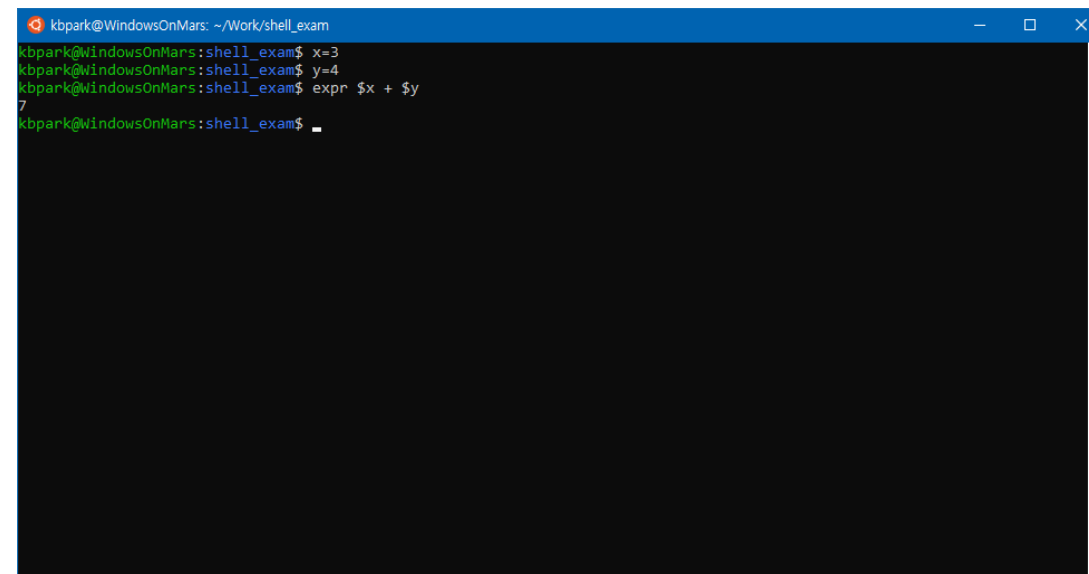
A terminal window titled 'kbpark@WindowsOnMars: ~/Work/shell_exam' with a blue header bar. The terminal shows the execution of a shell script. The prompt is 'kbpark@WindowsOnMars:shell_exam\$'. The command 'bash ./expr1.sh' is entered. The output is 'Total value : 4'. The prompt returns to 'kbpark@WindowsOnMars:shell_exam\$' with a cursor. The terminal has a black background and green text.

```
kbpark@WindowsOnMars: ~/Work/shell_exam
kbpark@WindowsOnMars:shell_exam$ bash ./expr1.sh
Total value : 4
kbpark@WindowsOnMars:shell_exam$ _
```

LINUX - SHELL SCRIPT

expr 명령어

- `x=3`
- `y=4`
- `expr $x + $y`

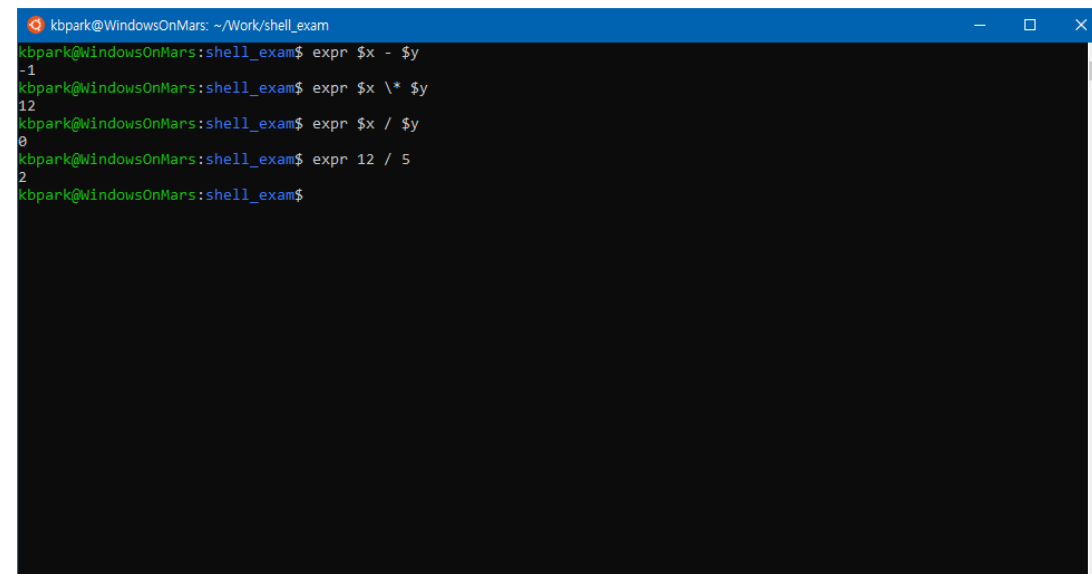
A terminal window titled 'kbpark@WindowsOnMars: ~/Work/shell_exam' with a blue header bar. The terminal shows a sequence of commands and their outputs: 'x=3' is entered and executed; 'y=4' is entered and executed; 'expr \$x + \$y' is entered and executed, resulting in the output '7'. The prompt 'kbpark@WindowsOnMars:shell_exam\$' is visible at the end of each line.

```
kbpark@WindowsOnMars: ~/Work/shell_exam
kbpark@WindowsOnMars:shell_exam$ x=3
kbpark@WindowsOnMars:shell_exam$ y=4
kbpark@WindowsOnMars:shell_exam$ expr $x + $y
7
kbpark@WindowsOnMars:shell_exam$
```

LINUX - SHELL SCRIPT

expr 명령어

- `expr $x - $y`
- `expr $x % $y`
- `expr $x / $y`
- `# -1`
- `# 12`
- `# 0`
- `expr 12 / 5`
- `# 2`

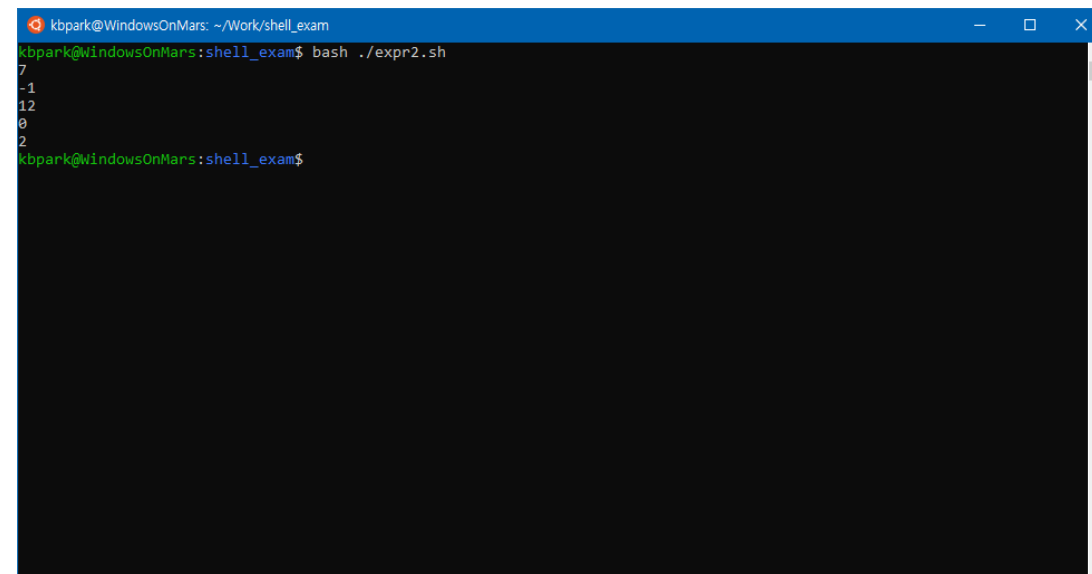
A terminal window titled 'kbpark@WindowsOnMars: ~/Work/shell_exam' showing the execution of the 'expr' command. The commands and their outputs are: 'expr \$x - \$y' returns '-1', 'expr \$x % \$y' returns '12', 'expr \$x / \$y' returns '0', and 'expr 12 / 5' returns '2'.

```
kbpark@WindowsOnMars: ~/Work/shell_exam
kbpark@WindowsOnMars:shell_exam$ expr $x - $y
-1
kbpark@WindowsOnMars:shell_exam$ expr $x % $y
12
kbpark@WindowsOnMars:shell_exam$ expr $x / $y
0
kbpark@WindowsOnMars:shell_exam$ expr 12 / 5
2
kbpark@WindowsOnMars:shell_exam$
```

LINUX - SHELL SCRIPT

expr 명령어

- `expr $x - $y`
- `expr $x % $y`
- `expr $x / $y`
- `# -1`
- `# 12`
- `# 0`
- `expr 12 / 5`
- `# 2`

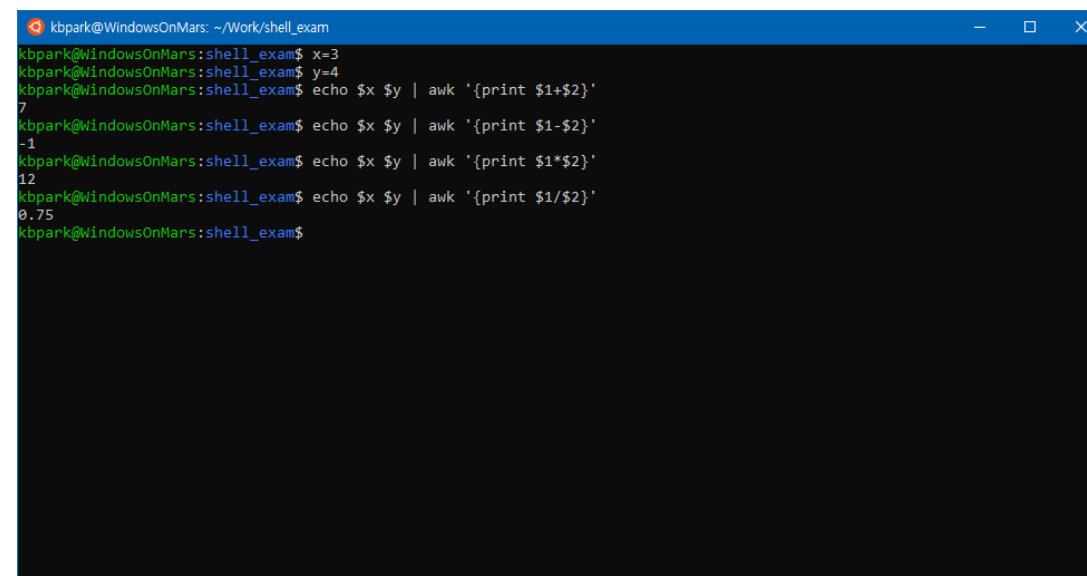


```
kbpark@WindowsOnMars: ~/Work/shell_exam
kbpark@WindowsOnMars:shell_exam$ bash ./expr2.sh
7
-1
12
0
2
kbpark@WindowsOnMars:shell_exam$
```

LINUX - SHELL SCRIPT

awk 명령어

- 리눅스에서 기본적으로 제공하는 명령어로 소수점 계산이 가능하다.
- 단 직접 변수를 사용하지 못하기 때문에 파이프를 통해 전달하는 과정이 필요하다.
 - `x=3`
 - `y=4`
 - `echo $x $y | awk '{print $1+$2}'`
 - `# 7`
 - `echo $x $y | awk '{print $1-$2}'`
 - `# -1`
 - `echo $x $y | awk '{print $1*$2}'`
 - `# 12`
 - `echo $x $y | awk '{print $1/$2}'`
 - `# 0.75`



```
kbpark@WindowsOnMars: ~/Work/shell_exam
kbpark@WindowsOnMars:shell_exam$ x=3
kbpark@WindowsOnMars:shell_exam$ y=4
kbpark@WindowsOnMars:shell_exam$ echo $x $y | awk '{print $1+$2}'
7
kbpark@WindowsOnMars:shell_exam$ echo $x $y | awk '{print $1-$2}'
-1
kbpark@WindowsOnMars:shell_exam$ echo $x $y | awk '{print $1*$2}'
12
kbpark@WindowsOnMars:shell_exam$ echo $x $y | awk '{print $1/$2}'
0.75
kbpark@WindowsOnMars:shell_exam$
```

LINUX - SHELL SCRIPT

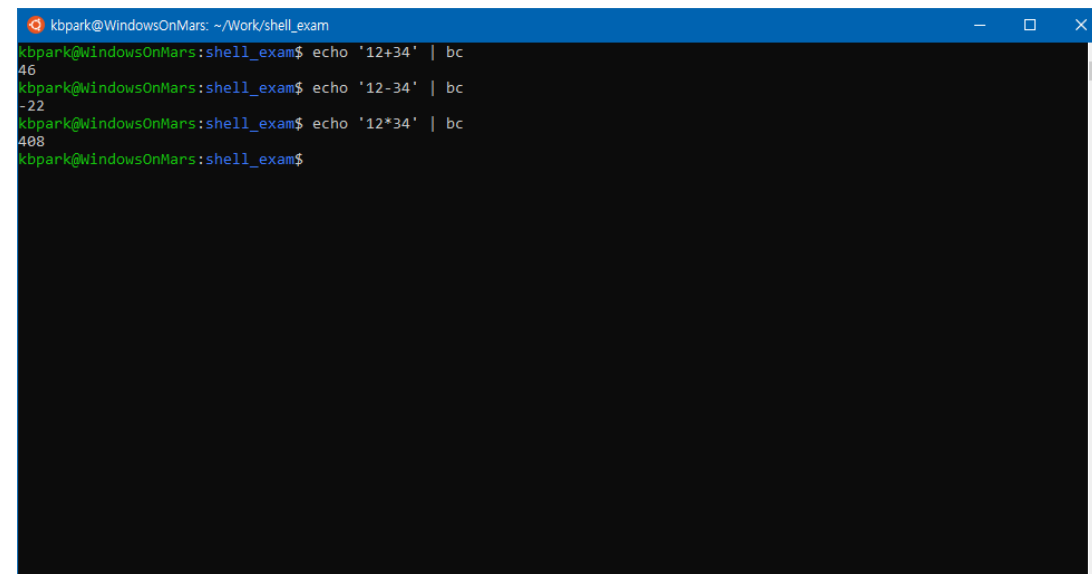
bc 명령어

- basic calculator의 약자로 리눅스 bc가 설치되어야 한다.
- 인터랙티브 모드와 배치 모드 둘다 사용이 가능하고, 실수, 사칙연산, 거듭제곱 등의 연산과 같은 고급 기능이 있으며 가법다는 특징이 있다.

LINUX - SHELL SCRIPT

bc 명령어

- `$ echo '12+34' | bc`
- 46
- `$ echo '12-34' | bc`
- -22
- `$ echo '12*34' | bc`
- 408

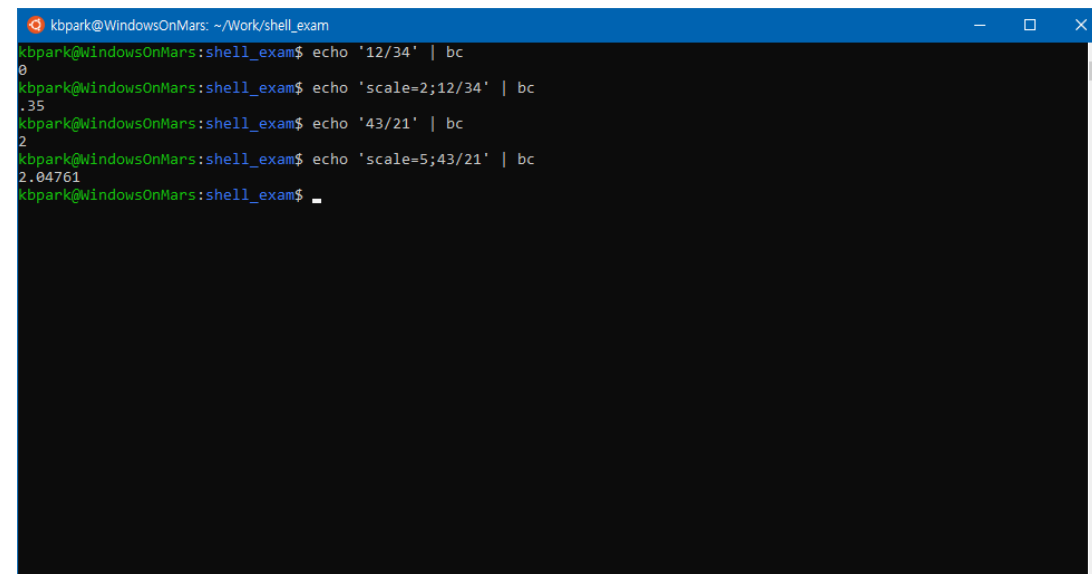
A terminal window titled 'kbpark@WindowsOnMars: ~/Work/shell_exam' showing three lines of shell commands and their outputs. The first line is 'kbpark@WindowsOnMars:shell_exam\$ echo '12+34' | bc' with output '46'. The second line is 'kbpark@WindowsOnMars:shell_exam\$ echo '12-34' | bc' with output '-22'. The third line is 'kbpark@WindowsOnMars:shell_exam\$ echo '12*34' | bc' with output '408'. The prompt 'kbpark@WindowsOnMars:shell_exam\$' is visible at the bottom.

```
kbpark@WindowsOnMars: ~/Work/shell_exam
kbpark@WindowsOnMars:shell_exam$ echo '12+34' | bc
46
kbpark@WindowsOnMars:shell_exam$ echo '12-34' | bc
-22
kbpark@WindowsOnMars:shell_exam$ echo '12*34' | bc
408
kbpark@WindowsOnMars:shell_exam$
```

LINUX - SHELL SCRIPT

bc 명령어

- `$ echo '12/34' | bc`
- 0
- `$ echo 'scale=2;12/34' | bc`
- .35
- `$ echo '43/21' | bc`
- 2
- `$ echo 'scale=5;43/21' | bc`
- 2.04761

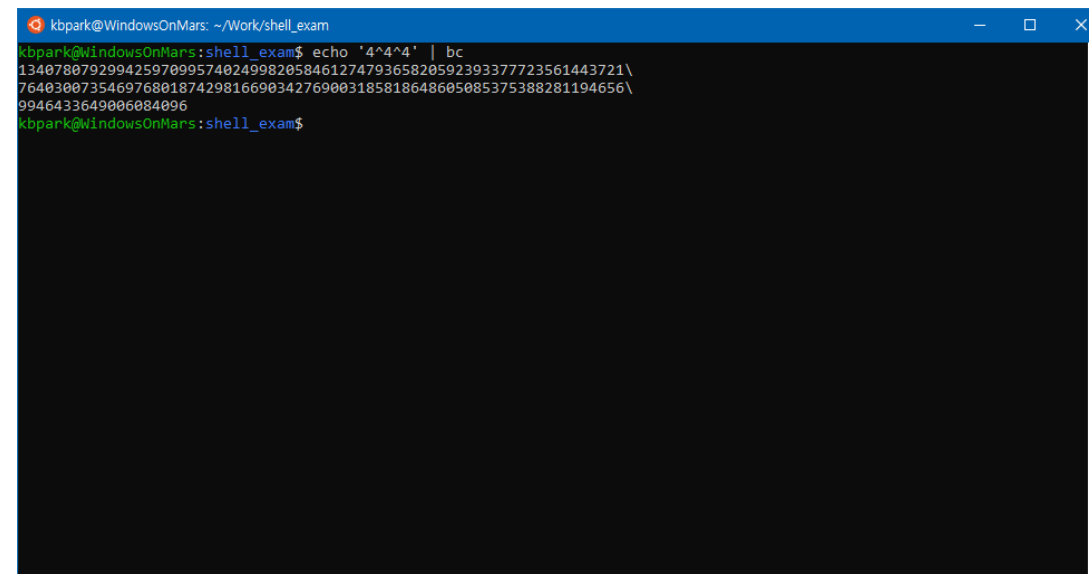
A terminal window titled 'kbpark@WindowsOnMars: ~/Work/shell_exam' showing the execution of several 'bc' commands. The commands and their outputs are: 'echo '12/34' | bc' outputs '0'; 'echo 'scale=2;12/34' | bc' outputs '.35'; 'echo '43/21' | bc' outputs '2'; and 'echo 'scale=5;43/21' | bc' outputs '2.04761'. The prompt 'kbpark@WindowsOnMars:shell_exam\$' is visible at the end of each line.

```
kbpark@WindowsOnMars: ~/Work/shell_exam
kbpark@WindowsOnMars:shell_exam$ echo '12/34' | bc
0
kbpark@WindowsOnMars:shell_exam$ echo 'scale=2;12/34' | bc
.35
kbpark@WindowsOnMars:shell_exam$ echo '43/21' | bc
2
kbpark@WindowsOnMars:shell_exam$ echo 'scale=5;43/21' | bc
2.04761
kbpark@WindowsOnMars:shell_exam$
```

LINUX - SHELL SCRIPT

bc 명령어

- `$ echo '4^4^4' | bc`
 - 13407807929942597099574024998205846127479365
820592393377723561443721W
76403007354697680187429816690342769003185818648
605085375388281194656W
9946433649006084096



```
kbpark@WindowsOnMars: ~/Work/shell_exam
kbpark@WindowsOnMars:shell_exam$ echo '4^4^4' | bc
13407807929942597099574024998205846127479365820592393377723561443721\
76403007354697680187429816690342769003185818648605085375388281194656\
9946433649006084096
kbpark@WindowsOnMars:shell_exam$
```

LINUX - SHELL SCRIPT

산술 연산자

Operator	Description	Example
+ (Addition)	덧셈	expr \$a + \$b
- (Subtraction)	뺄셈	expr \$a - \$b
* (Multiplication)	곱셈	expr \$a * \$b
/ (Division)	나누기	expr \$b / \$a
% (Modulus)	나머지 반환	expr \$b % \$a
= (Assignment)	값 할당	a = \$b
== (Equality)	비교	[\$a == \$b]
!= (Not Equality)	비교	[\$a != \$b]

관계 연산자

Operator	Description	Example
-eq	두 피연산자의 값이 동일한 여부를 확인하여 같으면 참을 반환한다.	[\$a -eq \$b]
-ne	두 피연산자의 값이 동일한지 여부를 확인하여 같지 않으면 참을 반환한다.	[\$a -ne \$b]
-gt	왼쪽 피연산자의 값이 오른쪽 피연산자의 값보다 큰지 확인하고, 그렇다면 조건이 참이된다.	[\$a -gt \$b]
-lt	왼쪽 피연산자의 값이 오른쪽 피연산자의 값보다 작은지 확인하고, 그렇다면 조건이 참이된다.	[\$a -lt \$b]
-ge	왼쪽 피연산자의 값이 오른쪽 피연산자의 값보다 크거나 같은지 확인하고, 그렇다면 조건이 참이된다.	[\$a -ge \$b]
-le	왼쪽 피연산자의 값이 오른쪽 피연산자의 값보다 작거나 같은지 확인하고, 그렇다면 조건이 참이된다.	[\$a -le \$b]

bash는 숫자 값과 관련된 관계 연산을 제공하며, 문자열은 제공되지 않는다.

LINUX - SHELL SCRIPT

Boolean 연산자

Operator	Description	Example
!	논리 부정	[! false] is true.
-o	OR 연산자	[\$a -lt 20 -o \$b -gt 100] is true.
-a	AND 연산자	[\$a -lt 20 -a \$b -gt 100] is false.

a가 10이고 b가 20으로 가정한다.

문자열 연산자

Operator	Description	Example
=	두 피연산자의 값을 확인하고 같다면 참이다.	[\$a = \$b]
!=	두 피연산자의 값을 확인하고 같지 않으면 참이다.	[\$a != \$b]
-z	주어진 문자열 피연산자의 길이가 0이면 참이다.	[-z \$a]
-n	주어진 문자열 피연산자의 0이 아니면 참이다.	[-n \$a]
str	빈 문자열인지를 확인하고 비어있으면 거짓을 반환한다.	[\$a]

bash에서 문자열 연산을 위해서는 다음과 같은 명령을 수행해야 한다.
변수 a에는 "abc"가 있고 변수 b에는 "efg"가 있다고 가정한다.

LINUX - SHELL SCRIPT

파일 테스트 연산자

- 파일과 관련된 속성을 테스트 하는데 확인하는 연산자이다.
- 파일의 이름은 test이고 크기가 100Byte, R/W/X 권한이 있다고 가정한다.

Operator	Description	Example
-b file	파일이 블록 파일인지 확인한다. (블록 디바이스 등)	[-b \$file]
-c file	파일이 문자 파일인지 확인한다. (키보드, 모뎀, 사운드 카드 등)	[-c \$file]
-d file	파일이 디렉터리인지 확인한다.	[-d \$file]
-f file	파일이 디렉터리나 일반파일인지 확인한다. (장치 파일이 아님)	[-f \$file]
-g file	파일에 SGID (Set Group ID)가 설정되어 있는지 확인한다.	[-g \$file]
-k file	파일에 고정 비트가 설정되어 있는지 확인한다.	[-k \$file]
-p file	파일이 명명 파이프인지 확인한다.	[-p \$file]
-t file	파일 설명자가 있고 터미널과 연결되어 있는지 확인한다.	[-t \$file]
-u file	파일에 SUID (Set User ID) 비트가 설정되어 있는지 확인한다.	[-u \$file]
-r file	파일을 읽을 수 있는지 확인한다.	[-r \$file]
-w file	파일 쓰기가 가능한지 확인한다.	[-w \$file]
-x file	파일 실행이 가능한지 확인한다.	[-x \$file]
-s file	파일의 크기가 0보다 큰지 확인한다.	[-s \$file]
-e file	파일이 존재하는지 확인한다.	[-e \$file]

LINUX - SHELL SCRIPT

연산자 - Shell Decision Making

- 이번에는 Unix Shell에서 특정 조건일 때, 올바른 수행이 가능하도록 하는 조건문에 대해서 알아본다.

The if...else statements

- if else 문은 주어진 옵션 집합에서 조건을 선택할 수 있도록 지원한다.
- 어떠한 조건에 대해서 True가 될 때 지정된 문이 실행되고, False일 경우 실행되지 않는다.
- 대부분 비교 연산자를 통해 작성한다.
- 각 구문에 대한 공백을 지켜야 오류가 발생하지 않는다.

LINUX - SHELL SCRIPT

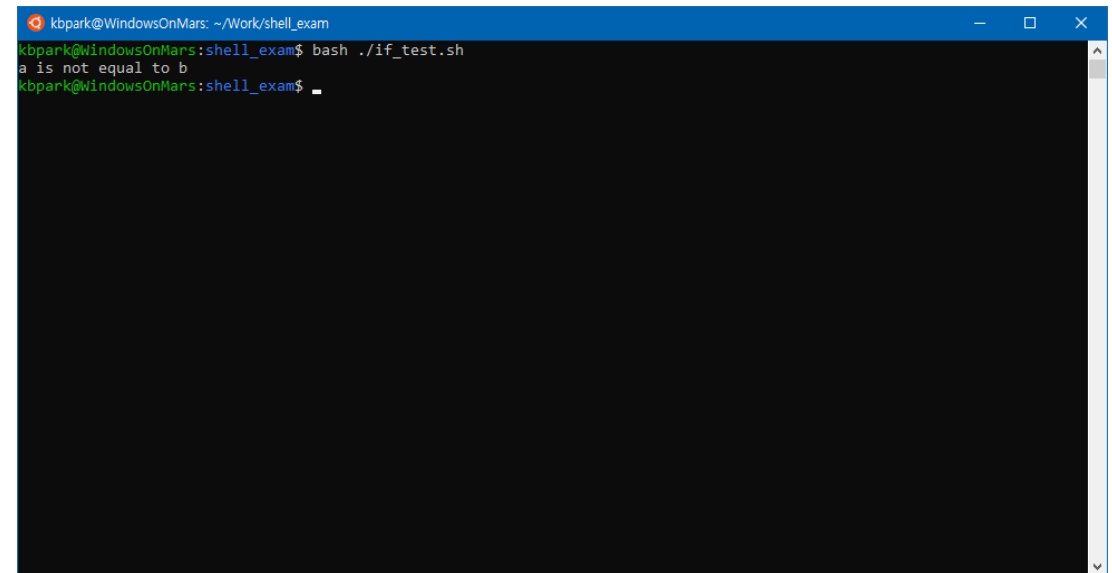
if...fi statement

- 문법
 - if [expression]
 - then
 - Statement(s) to be executed if expression is true
 - fi

LINUX - SHELL SCRIPT

if...fi statement

- `#!/bin/bash`
- `a=10`
- `b=20`
- `if [$a == $b]`
- `then`
- `echo "a is equal to b"`
- `fi`
- `if [$a != $b]`
- `then`
- `echo "a is not equal to b"`
- `fi`

A terminal window titled 'kbpark@WindowsOnMars: ~/Work/shell_exam' showing the execution of a shell script. The prompt is 'kbpark@WindowsOnMars:shell_exam\$'. The user enters 'bash ./if_test.sh'. The output is 'a is not equal to b'. The prompt returns to 'kbpark@WindowsOnMars:shell_exam\$' with a cursor.

```
kbpark@WindowsOnMars: ~/Work/shell_exam
kbpark@WindowsOnMars:shell_exam$ bash ./if_test.sh
a is not equal to b
kbpark@WindowsOnMars:shell_exam$ _
```