

# 셸 스크립트 (Shell Script)

동의과학대학교  
컴퓨터정보과

# 실습 1

Yesterday  
All my troubles seemed so far away  
Now it looks as though they're here to stay  
Oh, I believe in yesterday  
Suddenly  
I'm not half the man I used to be  
There's a shadow hanging over me  
Oh, yesterday came suddenly

Why she had to go,  
I don't know She wouldn't say  
I said something wrong  
Now I long for yesterday

Yesterday  
Love was such an easy game to play  
Now I need a place to hide away  
Oh, I believe in yesterday

Why she had to go,  
I don't know She wouldn't say  
I said something wrong  
Now I long for yesterday

Yesterday  
Love was such an easy game to play  
Now I need a place to hide away  
Oh, I believe in yesterday

yesterday.txt

# 실습 2 : Ubuntu에 GCC 컴파일러 설치 및 사용법

- GCC 설치  
\$ sudo apt-get install gcc
- GCC 설치 확인  
\$ gcc -version
- C 프로그램 컴파일 하기  
\$ gcc <file\_name> -o <name\_of\_executable>  
  
\$ gcc hello\_world.c -o hello\_world

※ apt-get  
시스템에서 사용 가능한 패키지 설치, 검색, 업데이트 등 수행, sudo로 실행

※ return 0;  
정상적으로 함수를 끝내고, 운영체제에 값을 반환, 현재 실행 중인 해당 함수를 벗어남

- 컴파일하고 실행 결과 확인해 보자  
// hello.c

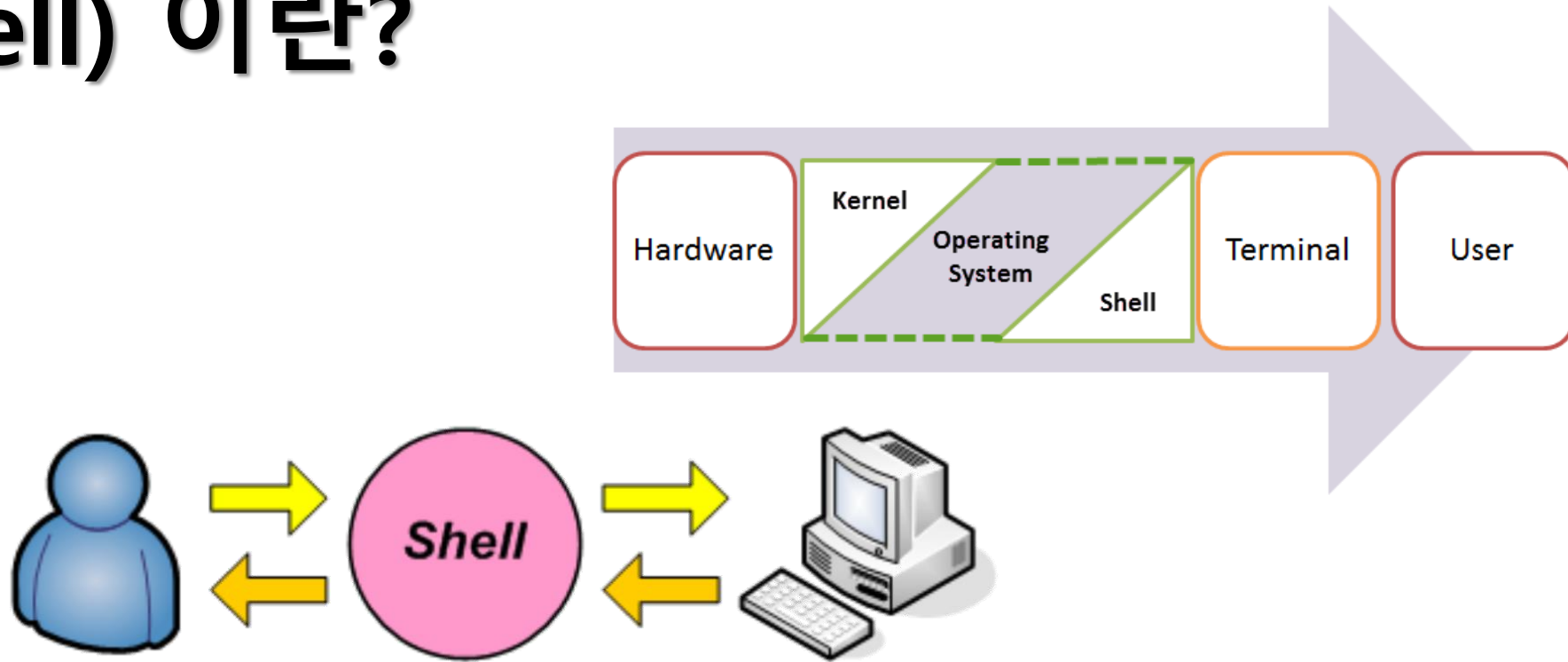
```
#include <stdio.h>

int main()
{
    int num;
    num = 1;
    printf("hello world!");
    printf("num = %d", num);

    return 0;
}
```

\$ gcc hello.c -o hello      컴파일  
\$ ./hello      실행

# 셸(Shell) 이란?



- **Shell** : 운영체제(Operating System)와 사용자 사이를 이어주는 역할을 하며, 사용자 명령어를 해석하고 실행한다.
- **Operating System** : 커널 위에 여러 가지 레이어를 탑재한 것으로 Shell 명령을 해석한다.
- **kernel** : 리눅스 운영체제의 핵심으로서 프로세스(process) 관리, 메모리 관리, I/O 시스템 관리, 파일 시스템 관리 등을 수행

# 스크립트(Script)란?

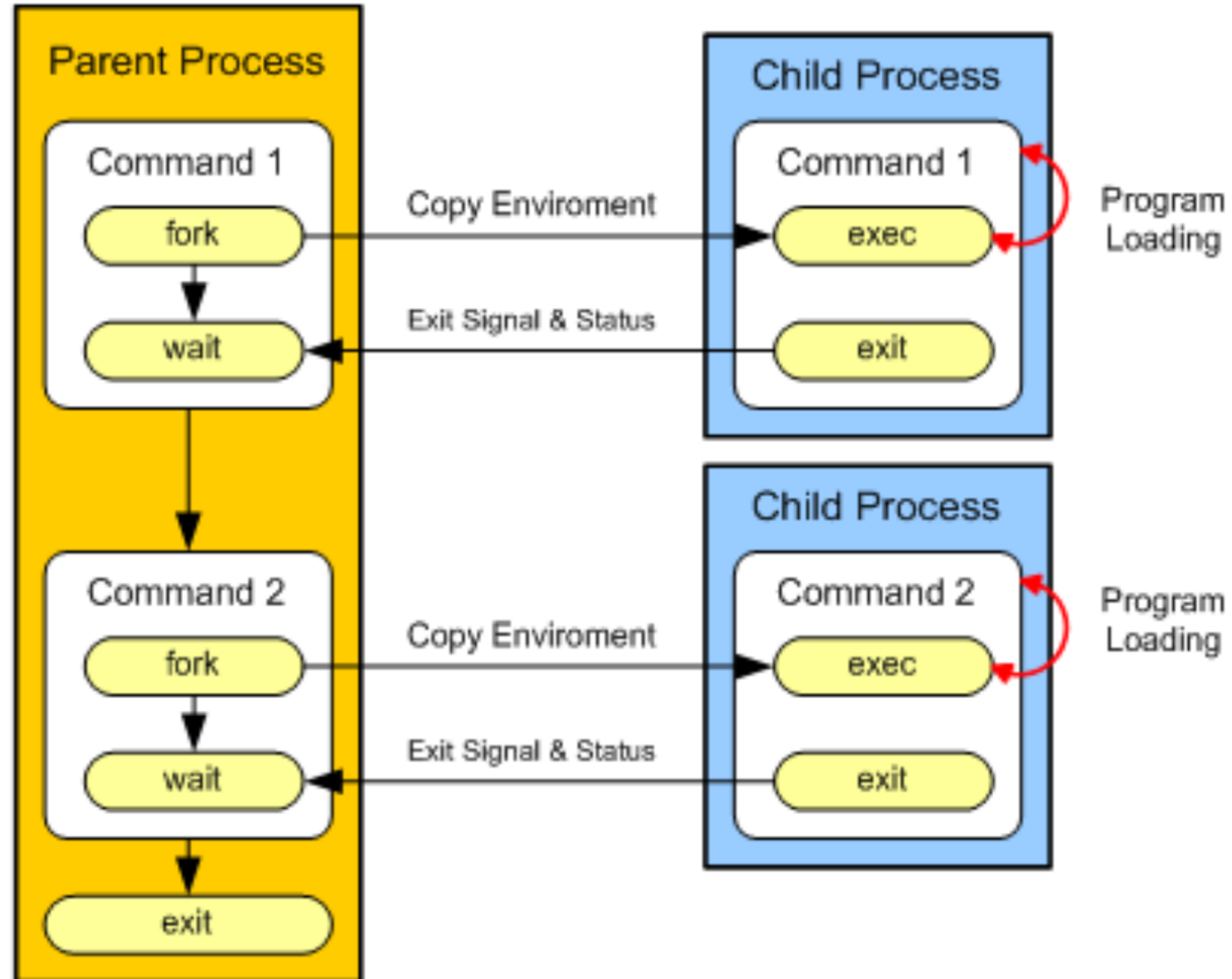
- 한번에 한 줄씩 읽어서 실행하는 프로그램
- 대화식으로 프로그램 가능
- 스크립트 언어로는 javascript, PHP, Perl 등

- ```
jkhim3217@Melon-HP-Elite: ~  
jkhim3217: $ ls -x /bin  
bash  
btrfs-image  
btrfsck  
bzip2  
bzip2recover  
chac1  
chvt  
date  
dmesg  
echo  
fgconsole  
fuser  
gunzip  
btrfs  
btrfs-map-logical  
btrfstune  
bzcmp  
bzfgrep  
bzless  
chgrp  
cp  
dd  
dnsdomainname  
ed  
fgrep  
fusermount  
gzexe  
btrfs-debug-tree  
btrfs-select-super  
bunzip2  
bzdifff  
bzgrep  
bzmooore  
chmod  
cpio  
df  
domainname  
egrep  
findmnt  
getfacl  
gzip  
btrfs-find-root  
btrfs-zero-log  
busybox  
bzegrep  
bzip2  
cat  
chown  
dash  
dir  
dumpkeys  
false  
fsck.btrfs  
grep  
hostname
```

# 웹 스크립트를 사용하는 이유

- 동일한 작업을 반복하여 수행하고자 할 때 **자동화**할 수 있다.
- 자신만의 강력한 유틸리티를 만들 수 있다.
- 관리자 작업을 커스터마이징 할 수 있다.
- 서비스 환경설정, 사용자 추가 같은 작업에 오류를 줄일 수 있다.
- 사용 예 :
  - 데이터 백업, 시스템 모니터링, 시스템 경고 메시지 발송, 사용자 관리, 메모리 관리 등

# 셸 스크립트 life cycle





# Shell의 종류

- bash (Bourne-Again Shell)
  - # 프롬프트 사용
  - /bin/bash
  - 가장 많이 사용되며, 최초 개발된 shell인 Bourne shell의 변형
- sh (Bourne Shell)
  - \$ 프롬프트 사용
  - 상호대화형 방식을 취하고 있지 않은 단점, 모든 UNIX 시스템 표준 구성 요소
- csh (C style Shell)
  - % 프롬프트 사용
  - 상호대화형 인터페이스 사용, Bourne Shell보다 강력함
- zsh
  - % 프롬프트 사용
  - bash shell과 비슷하게 동작하는 강력한 shell, 추가기능들과 테마 등 플러그인 제공

# 셸 스크립트의 시작

- vi 편집기 등을 이용하여 스크립트 파일 작성
- 스크립트 파일의 가장 첫 라인은 "**#!/bin/bash**"로 시작
  - bash로 작성되어 실행된다는 것을 의미 함
  - Bourne Shell의 경우 "**#!/bin/sh**"
- 셸 스크립트 변수 특징
  - 대소문자 구별
  - 미리 선언할 필요가 없음
  - 정의는 [변수명]=[값] = 기호 앞뒤로 공백이 없어야 함
  - 변수 사용은 \${변수명}
- 주석은 # 사용
- 출력은 echo 명령 사용
  - 줄 바꿈 문자가 자동으로 붙어서 출력됨

# 셸 스크립트 실행

- 실습 01 : 출력(echo)

**\$vi ex01.sh**

```
1 #! /bin/bash
2
3 echo "bash script test!!"
```

# 셸 스크립트 실행

- 셸 스크립트 실행 모드 변경
  - **\$ chmod +x ex01.sh**

```
jhkim3217:~/ShellTest$ ls -l
total 0
-rw-rw-rw- 1 jhkim3217 jhkim3217 40 Oct  1 10:31 ex01.sh
jhkim3217:~/ShellTest$ chmod +x ex01.sh
jhkim3217:~/ShellTest$ ls -l
total 0
-rwxrwxrwx 1 jhkim3217 jhkim3217 40 Oct  1 10:31 ex01.sh
jhkim3217:~/ShellTest$
```

- 셸 스크립트 실행
  - **\$ bash ./ex01.sh**

```
jhkim3217:~/ShellTest$ sh ./ex01.sh
bash script test!!
jhkim3217:~/ShellTest$
```

# 실습

- 아래의 절차를 스크립트(myscript.sh, vi 사용)로 만들고 실행하시오.
  - todayScript라는 이름의 하위 디렉토리를 만들기
  - 하위 디렉토리에 myTest1.txt myTest2.txt파일 만들기
  - myTest1.txt 파일을 복사하여 myTest3.txt로 붙여 넣기
  - 전체 폴더를 자세히 보기 하기

=> 스크립트??

# 환경변수

- **환경변수란?**

- 운영체제에서 특정 프로세스를 실행시키기위해 참조하는 변수
- 예를 들어 PATH 변수는 운영체제가 어떤 프로세스를 실행시킬 때, 경로를 찾는데 이용

- **환경변수 확인 명령:**

- echo : 특정 환경 변수 확인 -> \$ echo \$SHELL
- export : 환경 변수 값 설정  
\$export SHELL=/bin/sh  
\$echo \$SHELL  
\$export SHELL=/bin/bash  
\$echo \$SHELL
- export, env, printenv

- **환경변수 종류**

- \$HOME : 사용자의 home directory 경로
- \$SHELL : 사용자가 로그인 할 때 실행할 shell 경로
- \$PATH : 실행 파일을 찾을 경로를 저장
  - 사용자가 만든 실행 파일을 모아 둔 경로를 추가하면 편하게 실행 가능
  - 일시적으로 추가할 수 있으나 .bash\_profile 등의 자동실행 파일에 지정해 두면 편리

# 환경 변수 확인 실습 : env\_variable.sh

- \$ vi env\_variable.sh
- \$ chmod +x env\_variable.sh
- \$ ./env\_variable.sh

echo "HOSTNAME=\$HOSTNAME" #호스트 이름

echo "HOME=\$HOME" #사용자 홈 디렉토리

echo "LANG=\$LANG" #언어

echo "PATH=\$PATH" #설정된 경로

echo "SHELL=\$SHELL" #로긴 셸

echo "UID=\$UID" #사용자 아이디(숫자)

echo "USER=\$USER" #사용자 계정

# 변수 및 출력 : \$a, \$b, \$c

- 실습 02 : 변수 출력

```
1 #! /bin/bash
2
3 a=5
4 b=10
5 c="hello, bash"
6 echo "a=$a"
7 echo "b=$b"
8 echo "c=$c"
```

```
~
~
```

```
jhkim3217:~/ShellTest$
jhkim3217:~/ShellTest$ vi ex02.sh
jhkim3217:~/ShellTest$ sh ./ex02.sh
a=5
b=10
c=hello, bash
```

## 참고사항

- 변수는 변수명 앞에 \$표시함
- 변수에 대입된 값은 모두 문자열로 취급
- 변수에 들어 있는 값을 숫자로 해서 **+, -, \*, / 등의 연산을 하려면 `expr`을 사용**
- 수식에 괄호 또는 곱하기(\*)는 그 앞에 꼭 역슬래쉬(\) 붙임



# 연산자

- 산술 연산자: +, -, \*, /, %
  - \* 앞에는 escape 문자인 \를 붙이거나 따옴표(" ")로 감싸야 함
- 논리 연산자: | (or), & (and)
  - escape 문자인 \를 붙이거나 따옴표로 감싸야 함
- `expr 명령` 사용
  - 피연산자와 연산자 사이에는 공백이 있어야 함

```
expr 5 + 2  
expr 5 - 2  
expr 5 "*" 2  
expr 5 / 2  
expr 5 % 2  
expr 1 \| 0  
expr 1 "&" 0
```

```
7  
3  
10  
2  
1  
1  
0
```

# 값 입력 받기 : read

- 실습 03 : 입력
  - “echo -n” : [Enter] 키(No new line)를 입력하기를 기다림

```
#!/bin/bash
echo -n "input something : "
read input
echo "your input : $input"
~
~
```

```
jhkim3217:~/ShellTest$ sh ./ex03.sh
input something : happy
your input : happy
```

# expr 연산자를 이용한 사칙연산

- expr 연산자 사용 시 주의사항
  - 숫자 계산에 사용
  - 역꺾테이션(`) 사용 : **expr ....**
  - 곱셈 연산자 \*와 괄호() 앞에는 역슬래쉬(\)를 사용
  - 변수명과 데이터값 사이에 공백이 없어야 함
  - 모든 연산자와 숫자, 변수, 기호 사이에는 **공백**이 존재해야 함
- 실습 04 : expr



```
1 #!/bin/bash
2
3 a=1
4 b=2
5 echo "a=$a"
6 echo "b=$b"
7
8 result1=`expr $a + $b`
9 echo "a+b=$result1"
10
11 result2=`expr $a - $b`
12 echo "a-b=$result2"
13
14 result3=`expr $a \* $b`
15 echo "a*b=$result3"
16
17 result4=`expr $a / $b`
18 echo "a/b=$result4"
```

```
jhkim3217:~/ShellTest$ sh ./ex04.sh
a=1
b=2
a+b=3
a-b=-1
a*b=2
a/b=0
jhkim3217:~/ShellTest$
```

# 조건문 ( if, elif, else, fi)

- if

```
if condition
then
    command
fi
```

혹은

```
if condition ; then
    command
fi
```

- if-then-else

```
if condition
then
    command
else
    anothercommand
fi
```

- Nested if-then-else

```
if condition
then
    command
elif
    anothercommand
else
    yetanothercommand
fi
```

혹은

```
if condition1 ; then
    commands1
elif condition2 ; then
    commands2
fi
```

[ condition ]

공백

공백

- [ ]와 조건 사이에는 공백이 있어야함
- 조건의 끝을 알리는 ; (세미콜론)이 있어야함

기본 if 구문 (한 라인에 작성하는 방법)

```
if [ 조건 ]; then 명령문; fi
```

예)

```
if [[ -z $1 ]]; then echo "인수를 입력하세요"; fi
```

# 숫자 비교

- [  $A -gt B$  ] : A가 B보다 크다(**g**reater **t**han).
- [  $A -lt B$  ] : A가 B보다 작다(**l**ess **t**han).
- [  $A -ge B$  ] : A가 B보다 크거나 같다(**g**reater than or **e**qual to)
- [  $A -le B$  ] : A가 B보다 작거나 같다(**l**ess than or **e**qual to).
- [  $A -eq B$  ] : A와 B가 같다(**e**qual).
- [  $A -ne B$  ] : A와 B가 다르다(**n**ot **e**qual).

# 문자 비교

- [ "string1" = "string2" ] : 두 문자열이 같은 경우(==도 가능)
- [ "string1" ! "string2" ] : 두 문자열이 다른 경우(!=도 가능)
- [ -z "string" ] : 문자열의 길이가 0인 경우 true
- [ -n "string" ] : 문자열의 길이가 0이 아닌 경우 true

- 실습 05 : if-then-else      file: ex05.sh

```
1 #!/bin/bash
2
3 validName="DIT"
4
5 echo -n "Enter name:"
6 read name
7 █
8 if [ $name = $validName ]
9 then
10     echo "You entered CORRECT name!!"
11 else
12     echo "$name is INCORRECT!!"
13 fi
```

```
$ ex05.sh
Enter name:Korea
Korea is INCORRECT!!

$ ex05.sh
Enter name:DIT
You entered CORRECT name!!
```

## • 실습 05 : log 파일 백업 명령어 만들기

- 먼저 3개의 log 파일(a.log, b.log, c.log)을 현재 디렉터리(./)에 만든다.
- 쉘 프로그램을 실행하면 ./bak 디렉터리가 없으면 디렉토리를 만들고, 3개의 log 파일을 복사(cp) 한다.
- bak 디렉터리가 있으면 바로 log 파일을 ./bak에 복사

```
1 #!/bin/bash
2
3 if ! [ -d bak ]; then
4     mkdir bak
5 fi
6
7 cp *.log bak
8
```

디렉토리 : 파일검사 연산자

```
jhkim3217:~/ShellTest$ touch a.log
jhkim3217:~/ShellTest$ touch b.log
jhkim3217:~/ShellTest$ touch c.log
jhkim3217:~/ShellTest$ ls -x *.log
a.log b.log c.log
jhkim3217:~/ShellTest$ sh ./ex05-1.sh
jhkim3217:~/ShellTest$ ls -x bak
a.log b.log c.log
```

\* 참고 동영상 보기 : <https://opentutorials.org/course/2598/14204>



## • 파일 검사 연산자

| 제어문       | 설명                                   |
|-----------|--------------------------------------|
| <b>-a</b> | 파일이 존재한 경우 true                      |
| <b>-b</b> | 파일이 존재하고 블록장치 파일인 경우 true            |
| <b>-c</b> | 파일이 존재하고 캐릭터 장치 파일인 경우 true          |
| <b>-d</b> | 파일이 존재하고 디렉토리인 경우 true               |
| <b>-e</b> | 파일이 존재하고 파일이 있는 경우 true              |
| <b>-f</b> | 파일이 존재하고 정규 파일인 경우 true              |
| <b>-g</b> | 파일이 존재하고 SetGID가 설정된 경우 true         |
| <b>-h</b> | 파일이 존재하고 한 개 이상의 심볼릭 링크가 설정된 경우 true |
| <b>-k</b> | 파일이 존재하고 Sticky bit가 설정된 경우 true     |
| <b>-p</b> | 파일이 존재하고 FIFO인 경우 true               |
| <b>-r</b> | 파일이 존재하고 읽기 가능한 경우 true              |
| <b>-s</b> | 파일이 존재하고 0보다 큰 경우 true               |
| <b>-u</b> | 파일이 존재하고 SetUID가 설정된 경우 true         |
| <b>-w</b> | 파일이 존재하고 쓰기가 가능한 경우 true             |
| <b>-x</b> | 파일이 존재하고 실행 가능한 경우 true              |

# 반복문

- 반복문 내에 break 문, continue 문을 넣을 수 있다.

## while 문

```
while condition  
do  
    command  
done
```

## until 문

```
until condition  
do  
    command  
done
```

## for in 문

```
for item in list  
do  
    command  
done
```

- 실습 06 : 반복문(while)

```
#!/bin/bash
```

```
i=1
```

```
while [ $i -lt 4 ]
```

```
do
```

```
    echo $i
```

```
    i=$((i+1))
```

```
done
```

**$\$(($  연산식  $)$**

```
jhkim3217:~/ShellTest$ sh ./ex06.sh
```

```
1
```

```
2
```

```
3
```

- **\$(( 연산식 ))** 사용
  - 연산자 앞뒤 공백에 상관없음
  - 출력을 위해 echo 사용
- 연산 결과를 변수에 저장 방법
  - ESC 키 밑에 있는 ` (역꺾테이션)으로 표현식을 묶으면 그 표현식을 먼저 처리하라는 의미
  - 예 : `ls -l `find . -name "*.sh"``

```
echo $((5 + 2))  
echo $((5 - 2))  
echo $((5 * 2))  
echo $((5 / 2))  
echo $((5 % 2))  
echo $((1 | 0))  
echo $((1 & 0))
```

```
7  
3  
10  
2  
1  
1  
0
```

```
let a=(5 + 2)  
b=$(expr 5 - 2)  
c=$((5 * 2))  
d=`expr 5 / 2`
```

```
echo $a  
echo $b  
echo $c  
echo $d
```

```
7  
3  
10  
2
```

# case 문

- \*) 는 명시적으로 지정한 값들 외의 모든 case를 처리함

## • 실습 07

```
case value in
  a)
    command
    #...
    ;;
  b)
    command
    #...
    ;;
esac
```

```
#!/bin/bash

read -p "당신은 신발을 몇켤레 가지고 있습니까?" value

case $value in
  0|1)
    echo "신발이 적어서 당신은 걷기 어렵다."
    ;;
  2)
    echo "적당하다. 걸으러 나가시오!"
    ;;
  *)
    echo "필요한 것보다 많은 신발을 가지고 있다."
    ;;
esac
```

```
jhkim3217:~/ShellTest$ sh ./ex07.sh
당신은 신발을 몇켤레 가지고 있습니까?1
신발이 적어서 당신은 걷기 어렵다.
jhkim3217:~/ShellTest$ sh ./ex07.sh
당신은 신발을 몇켤레 가지고 있습니까?4
필요한 것보다 많은 신발을 가지고 있다.
jhkim3217:~/ShellTest$
```

## • 과제 : 실습 08 : case 문을 이용한 사칙연산

```
1 #!/bin/bash
2
3 echo "Select the operation you want to run."
4 echo "1=plus, 2=minus, 3=Multiplication, 4=division"
5
6 read req
7
8 case "$req" in
9 1)
10 echo "enter a="
11 read a
12 echo "enter b="
13 read b
14 result1=`expr $a + $b`
15 echo "a+b=$result1" ;;
16
17 2)
18 echo "enter a="
19 read a
20 echo "enter b="
21 read b
22 result2=`expr $a - $b`
23 echo "a-b=$result2" ;;
24
25 3)
26 echo "enter a="
27 read a
28 echo "enter b="
29 read b
30 result3=`expr $a #* $b`
31 echo "a*b=$result3" ;;
32
33 4)
34 echo "enter a="
35 read a
36 echo "enter b="
37 read b
38 result4=`expr $a / $b`
39 echo "a/b=$result4" ;;
40
41 esac
42 exit 0
```

```
jhkim3217:~/ShellTest$ sh ./ex05.sh
Select the operation you want to run.
1=plus, 2=minus, 3=Multiplication, 4=division
1
enter a=
5
enter b=
3
a+b=8
jhkim3217:~/ShellTest$ !s
sh ./ex05.sh
Select the operation you want to run.
1=plus, 2=minus, 3=Multiplication, 4=division
2
enter a=
9
enter b=
3
a-b=6
jhkim3217:~/ShellTest$ !s
sh ./ex05.sh
Select the operation you want to run.
1=plus, 2=minus, 3=Multiplication, 4=division
3
enter a=
3
enter b=
5
a*b=15
jhkim3217:~/ShellTest$ !s
sh ./ex05.sh
Select the operation you want to run.
1=plus, 2=minus, 3=Multiplication, 4=division
4
enter a=
8
enter b=
2
a/b=4
```

- 실습 09 : bc 명령어를 이용한 계산기 작성

- /bin/bc : Linux 쉘 계산기

```
# bc
```

```
# echo 12+34 | bc
```

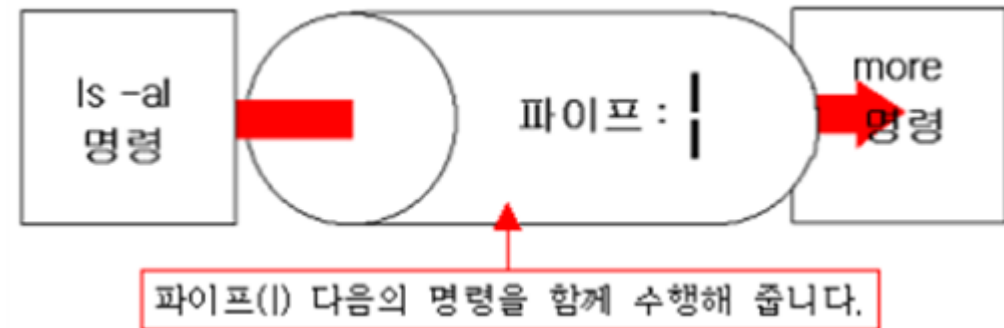
```
# echo 12-34 | bc
```

```
# echo 12*34 | bc
```

- pipe 명령어 : |

- \$ ls -al | more

- \$ cat test.txt | grep abc



```
1 #!/bin/bash
2
3 echo "-----"
4 echo "| shell script calculator |"
5 echo "-----"
6
7 while [ true ]
8 do
9     read A
10    echo `echo "$A"|bc`
11 done
12
```

```
jhkim3217:~/ShellTest$ sh ./ex09.sh
```

```
| shell script calculator |
```

```
(4+3)*7/7
```

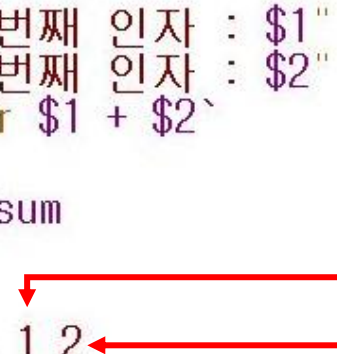
```
7
```

실행 멈추고 나가기 : Ctrl + C

# 함수

- 함수 선언 : 호출하기 전에 선언
- 함수 호출 : 함수명으로 호출
- 실습 10

```
1 #! /bin/bash
2
3 two_sum_fun()
4 {
5     echo "첫번째 인자 : $1"
6     echo "두번째 인자 : $2"
7     sum=`expr $1 + $2`
8
9     return $sum
10 }
11
12 two_sum_fun 1 2
13 result=$?
14
15 echo "합은 $result입니다."
```



```
jhkim3217:~/ShellTest$ sh ./ex10.sh
첫번째 인자 : 1
두번째 인자 : 2
합은 3입니다.
jhkim3217:~/ShellTest$
```



## • 위치 매개 변수

| 이름  | 의미                                                       |
|-----|----------------------------------------------------------|
| \$0 | 실행된 스크립트 이름                                              |
| \$1 | \$1 \$2 \$3...\${10}인자 순서대로 번호가 부여된다. 10번째부터는 "{}"감싸줘야 함 |
| \$* | 전체 인자 값                                                  |
| \$@ | 전체 인자 값(\$* 동일하지만 쌍따옴표로 변수를 감싸면 다른 결과 나옴)                |
| \$# | 매개 변수의 총 개수                                              |

## • 특수 매개 변수

| 이름   | 의미                              |
|------|---------------------------------|
| \$\$ | 현재 스크립트의 PID                    |
| \$?  | 최근에 실행된 명령어, 함수, 스크립트 자식의 종료 상태 |
| \$!  | 최근에 실행한 백그라운드(비동기) 명령의 PID      |
| \$-  | 현재 옵션 플래그                       |
| \$_  | 지난 명령의 마지막 인자로 설정된 특수 변수        |

```
#!/bin/bash
```

```
echo $$  
echo $0  
echo $1  
echo $*  
echo $#  
echo ""  
v=10  
echo $v
```

셸 명령 프롬프트

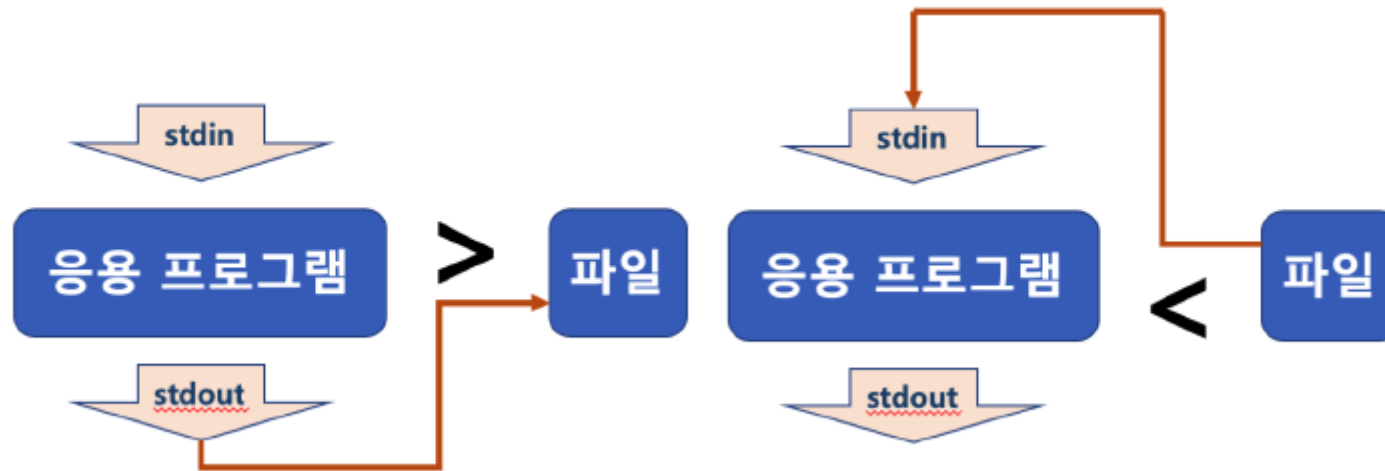
```
# ./test.sh a b c
```

실행결과

```
87  
./test.sh  
a  
a b c  
3  
  
10
```

# 파이프(pipe), 리다이렉션(redirection)

- 리다이렉션(redirection) : >, >>, <



# 파이프(pipe), 리다이렉션(redirection)

- 리다이렉션(redirection)

- 표준출력(덮어쓰기)

```
$ ls > ls_list.txt
$ ls
01.txt 02.txt ls_list.txt
$ cat ls_list.txt
01.txt
02.txt
ls_list.txt
```

- 표준출력(추가)

```
$ ls >> ls_list.txt
$ ls
01.txt 02.txt ls_list.txt
$ cat ls_list.txt
01.txt
02.txt
ls_list.txt
01.txt
02.txt
ls_list.txt
```

- 표준입력

```
$ tail -n 2 < ls_list.txt
02.txt
ls_list.txt
```

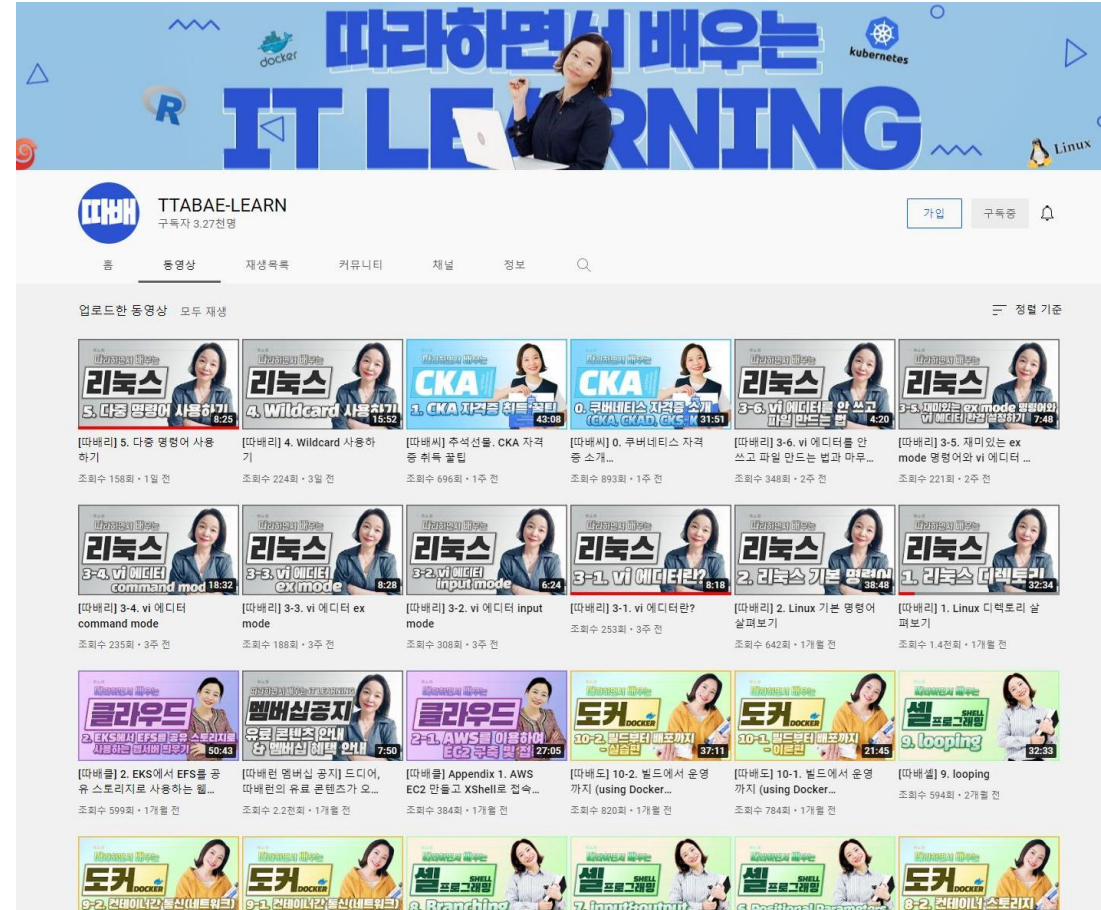
- 참고 : <https://bit.ly/3joVKcO>

## shell 메타 문자

| 문자    | 의미                              |
|-------|---------------------------------|
| >     | 표준 출력을 파일에 기록하는 출력 리다이렉션 기호     |
| >>    | 표준 출력을 파일의 끝에 추가하는 출력 리다이렉션 기호  |
| <     | 파일로부터 표준 입력을 읽어 들이는 입력 리다이렉션 기호 |
| *     | 널 문자열을 포함한 모든 문자열 치환            |
| ?     | 모든 단일 문자와 치환                    |
| [...] | 대괄호 안의 어떠한 문자와도 일치하는 파일 치환 대표문자 |
|       | 표준 출력을 입력으로 보내는 파이프 기호          |
|       | 조건부 실행 : 이전 명령이 실패하면 실행         |
| ;     | 명령어의 순서에 사용                     |
| &     | 백그라운드 모드 실행                     |
| &&    | 조건부 실행 : 이전 명령이 성공할 경우 실행       |
| #     | 주석 처리                           |
| \$    | 변수의 접근                          |

# Linux 공부 YouTube 좋은 강좌 추천

- [https://www.youtube.com/channel/UC\\_VOQjI7mtQTEaTXXQIzLtQ/videos?view=0&sort=dd&shelf\\_id=0](https://www.youtube.com/channel/UC_VOQjI7mtQTEaTXXQIzLtQ/videos?view=0&sort=dd&shelf_id=0)
- Linux 부분 시청 추천



# Shell 프로그래밍 YouTube 강좌 추천

- <https://www.youtube.com/watch?v=cXnVygkAg4I>
- <https://www.youtube.com/watch?v=HZfaBDM3EW0>