

## BASH Shell Script – 3rd Lab

### 1. 셸 스크립트

간단한 셸 스크립트

```
$ vi hello.sh

(hello.sh)
#!/bin/bash
echo hello world

$ chmod 755 hello.sh
$ ./hello.sh
hello world
```

- #! 은 셸에게 이 프로그램을 실행하기 위해서 #! 다음에 오는 아규먼트를 실행프로그램으로 사용한다는 것을 알려주기 위해서 사용된다.
- Unix/Linux system은 Windows와 달리 파일의 확장자명(.exe .com 등)으로 실행파일 유무를 결정하지 않으며, 파일권한 설정의 변경을 통하여 실행파일로 결정한다. 따라서 스크립트 파일을 실행하기 위해서는 파일에 실행권한을 주어야 한다.

### 2. 주석

- "#" 에서부터 라인의 마지막까지가 주석으로 처리된다.

```
#!/bin/bash
# 만든이 : comp-ta
# 하는일 : hello world 를 출력한다.

echo "hello world" # 문자열을 출력한다.
```

### 3. 변수

- C 언어와 같은 변수 선언을 필요로 하지 않는다.
- 기본적으로 데이터를 string(문자열)로 저장한다.(수치를 대입해도 실제 수치가 아닌 문자열이 저장된다.) 따라서 타입이 없고 아무 값이나 저장할 수 있다.
- 변수의 값을 사용할 때는 변수명앞에 "\$"를 붙여서 사용하고, 값을 대입할때는 "\$"를 붙이지 않는다.
- 변수에 데이터를 저장하기 위해서는 대입연산자 "=" 를 사용한다. 대입연산자와, 피연산자/연산자 사이에는 공백이 존재하면 안 된다.

```
varname=value
```

## 4190.102A Computer Programming (2007 Winter)

- 산술연산 : 모든 것이 문자열로 저장되기 때문에 산술연산을 하기 위해서는 변수=\$((산술식)) 과 같이 사용.

```
a=1
b=2
c=$((a+b))
echo $c
```

(3 출력)

### 4. 명령행 인자(argument)

- \$1~ \$n - 넘겨진 인자들
- \$\* - 스크립트에 전달된 인자들을 모아놓은 문자열. 하나의 변수에 저장되며 IFS 환경변수의 첫번째 문자로 구분된다. (IFS : internal field separator)
- @\$ - \$\*과 같다. 다만 구분자가 IFS변수의 영향을 받지 않는다.
- \$0 - 실행된 쉘 스크립트 이름
- \$# - 스크립트에 넘겨진 인자의 갯수

```
(test1.sh)
#!/bin/bash
echo "This Script Executable File : $0"
echo "Argument Count : $#"
```

```
echo "Argument List \$* : $*"
echo "Argument List \@ : \@"
```

```
echo "Argument 1 : $1"
echo "Argument 2 : $2"
echo "Argument 3 : $3"
echo "Argument 4 : $4"
```

(실행)

```
$chmod 755 test1.sh
$./test1.sh a1 a2 a3 a4
This Script Executable File : ./test1.sh
Argument Count : 4
Argument List $* : a1 a2 a3 a4
Argument List \@ : a1 a2 a3 a4
Argument 1 : a1
Argument 2 : a2
Argument 3 : a3
Argument 4 : a4
```

### 5. 제어구조

- if / else
  - 조건을 test하여 참이면 then부분을, 거짓이면 else부분을 실행.
  - test 조건들은 "[ ]" 사이에 쓰인다. "[" 과 "]" 사이에는 반드시 공백문자가 들어가야 한다.
  - 변수 값, 파일 특징, 명령의 실행 여부 등 다양한 test 조건이 있다.

4190.102A Computer Programming  
(2007 Winter)

Syntax	Example
<pre> <b>if</b> condition <b>then</b>     statements [<b>elif</b> condition <b>then</b>     statements...] [<b>else</b>     statements] <b>fi</b> </pre>	<pre> <b>if</b> [ \$1 -lt \$2 ] <b>then</b>     echo \$1 is less than \$2 <b>elif</b> [ \$1 -gt \$2 ] <b>then</b>     echo \$1 is greater than \$2 <b>else</b>     echo \$1 equals to \$2 <b>fi</b> </pre>

- 참고 : test 명령 조건

<p><u>문자열 비교</u></p> <pre> [ string ] - string이 빈 문자열이 아니라면 참 [ string1 = string2 ] - 두 문자열이 같다면 참 [ string1 != string2 ] - 두 문자열이 다르다면 참 [ -n string ] - 문자열이 null(빈 문자열) 이 아니라면 참 [ -z string ] - 문자열이 null(빈 문자열) 이라면 참 </pre>
<p><u>산술 비교</u></p> <pre> [ expr1 -eq expr2 ] - 두 표현식 값이 같다면 참 ('Equal') [ expr1 -ne expr2 ] - 두 표현식 값이 같지 않다면 참 ('Not Equal') [ expr1 -gt expr2 ] - expr1 &gt; expr2 이면 참 ('Greater Than') [ expr1 -ge expr2 ] - expr1 &gt;= expr2 이면 참 ('Greater Equal') [ expr1 -lt expr2 ] - expr1 &lt; expr2 이면 참 ('Less Than') [ expr1 -le expr2 ] - expr1 &lt;= expr2 이면 참 ('Less Equal') [ ! expr ] - expr 이 참이면 거짓, 거짓이면 참 [ expr1 -a expr2 ] - expr1 AND expr2 의 결과 (둘다 참이면 참, 'And') [ expr1 -o expr2 ] - expr1 OR expr2 의 결과 (둘중 하나만 참이면 참, 'Or') </pre>
<p><u>파일 조건</u></p> <pre> [ -b FILE ] - FILE 이 블록 디바이스 이면 참 [ -c FILE ] - FILE 이 문자 디바이스 이면 참. [ -d FILE ] - FILE 이 디렉토리이면 참 [ -e FILE ] - FILE 이 존재하면 참 [ -f FILE ] - FILE 이 존재하고 정규파일이면 참 [ -g FILE ] - FILE 이 set-group-id 파일이면 참 [ -h FILE ] - FILE 이 심볼릭 링크이면 참 [ -L FILE ] - FILE 이 심볼릭 링크이면 참 [ -k FILE ] - FILE 이 Sticky bit 가 셋팅되어 있으면 참 [ -p FILE ] - True if file is a named pipe. [ -r FILE ] - 현재 사용자가 읽을 수 있는 파일이면 참 [ -s FILE ] - 파일이 비어있지 않으면 참 [ -S FILE ] - 소켓 디바이스이면 참 [ -t FD ] - FD 가 열려진 터미널이면 참 [ -u FILE ] - FILE 이 set-user-id 파일이면 참 [ -w FILE ] - 현재 사용자가 쓸 수 있는 파일(writable file) 이면 참 [ -x FILE ] - 현재사용자가 실행할 수 있는 파일(Executable file) 이면 참 [ -O FILE ] - FILE 의 소유자가 현재 사용자이면 참 [ -G FILE ] - FILE 의 그룹이 현재 사용자의 그룹과 같으면 참 </pre>

4190.102A Computer Programming  
(2007 Winter)

[ FILE1 -nt FILE2 ] - : FILE1이 FILE2 보다 새로운 파일이면 ( 최근파일이면 ) 참
[ FILE1 -ot FILE2 ] - : FILE1이 FILE2 보다 오래된 파일이면 참
[ FILE1 -ef FILE2 ] - : FILE1 이 FILE2의 하드링크 파일이면 참

- for
  - 지정된 범위 안에서 루프를 수행한다. 범위는 어떤 집합도 가능하다.
  - (Bash 2.03버전 이후 버전) C style의 for loop 사용가능
    - 아래 3 문장은 모두 같은 의미이다 -
    - for a in 1 2 3 4 5 6 7 8 9 10
    - for a in \$(seq 1 10)
    - for ((a=0;a<10;a++))
  - 범위가 하나의 string이라면 IFS를 이용해 구분가능.
  - [in list]를 생략한다면 기본값은 "\$@"

Syntax	Example
<pre>for name [in list] do   statements that can use \$name. done</pre>	<pre>IFS=: for dir in \$PATH do   ls -ld \$dir done</pre>

- while
  - 어떤 조건이 참인 동안 코드를 반복

Syntax	Example
<pre>while condition do   statements... done</pre>	<pre>count=0 while [ \$count -lt 10 ] do   echo \$count   count=\$((count+1)) done</pre>

- until
  - 어떤 조건이 참일 때까지 (거짓인 동안) 반복

Syntax	Example
<pre>until condition do   statements... done</pre>	<pre>count=0 until [ \$count -gt 10 ] do   echo \$count   count=\$((count+1)) done</pre>

- case
  - C의 switch구문과 비슷한 역할을 함
  - C의 switch 구문은 정수나 문자 같은 단순한 값을 검사하는 반면, Bash의 case는

4190.102A Computer Programming  
(2007 Winter)

와일드카드 (?, \*) 사용이 허락되는 어떤 패턴을 놓고 문자열을 검사

Syntax	Example
<pre> <b>case</b> expression <b>in</b>   pattern1 )     statements ;;   pattern2 )     statements ;;   ... <b>esac</b> </pre>	<pre> <b>case</b> \$1 <b>in</b>   *.jpg )     echo jpg: Graphic   File!! ;;   *.txt )     echo txt: Text File!! ;;   * )     echo Unknown File!! ;; <b>esac</b> </pre>

➤ **select**

- Korn Shell, Bash Shell에만 존재
- 간단한 메뉴를 제공
- select의 루프 내에서는 자동적으로 루프를 벗어날 수 없기 때문에 반드시 break문등을 사용해서 루프를 벗어나야 한다.
- prompt string을 변경하기 위해서는 쉘변수 PS3를 이용.

Syntax
<pre> <b>select</b> name [<b>in</b> list]; <b>do</b>   statements that can use \$name...   ... <b>done</b> </pre>
Example
<pre> #!/bin/bash  echo "다음 중 scripting language 에 속하는 것은?" PS3="숫자를 선택하세요 : " <b>select</b> var <b>in</b> "Bash script" "C/C++" "Java" "Exit" <b>do</b>   <b>if</b> [ "\$var" = "Bash script" ]   <b>then</b>     echo "정답입니다."     break   <b>elif</b> [ "\$var" = "Exit" ]   <b>then</b>     echo "종료합니다."     exit 1   <b>else</b>     echo "\$var 을 선택하셨습니다. 오답입니다."     echo "다음 중 scripting language 에 속하는 것은?"   <b>fi</b> <b>done</b> </pre>

cf.) **read** : 직접 사용자 입력을 받음.

## 4190.102A Computer Programming (2007 Winter)

```
#!/bin/bash
echo "이름을 입력해주세요 : "
read NAME
echo "Hi $NAME!"
```

### 6. Quoting

#### ➤ Single Quotes

- Single Quotes 안의 모든 특수 문자를 보통 문자로 취급
- Single Quotes 안에 또 다른 Single Quote를 쓸 수 없음

```
$ echo 2 * 3 > 5 is a valid inequality. ....(X)
$ echo '2 * 3 > 5 is a valid inequality.' .....(O)
$ echo Hatter's tea party .....(X)
$ echo Hatter\'s tea party .....(O)
$ echo 'Hatter\'\'s tea party' .....(O)
```

#### ➤ Double Quotes

- Weak Single Quotes
- Single Quotes와 달리 \$, `(backquote), W(이스케이프)는 처리
- 문자열 안에서 다음을 참조할 때 사용
  - 변수확장 : \$varname
  - 명령 : \$(command) 혹은 `command` -single quote가 아니라 backquote임.
  - 산술연산구문 : \$((1+1))

```
$ echo "2 * 3 > 5 is a valid inequality."
$ echo "Hatter\'s tea party"

$ echo "path=$PATH"
$ echo "current directory=$(pwd)"
$ echo "current directory=`pwd`"
$ echo "1 + 1 = $((1+1))"
```

### 7. 함수

- 다른 프로그램 언어에서와 같이 코드를 재사용하거나 모듈화를 위해 함수를 사용
- 함수는 함수가 불리기 전에 정의되어야 한다. (C처럼 함수를 미리 선언하는 방법은 없음)
- parameter : 함수는 자신에게 넘어온 인자를 \$1, \$2와 같이 인자의 위치로 참조한다.
- return value : 함수는 반드시 정수값만 리턴할 수 있다. 이 리턴값은 \$? 변수에 저장된다.

**4190.102A Computer Programming  
(2007 Winter)**

```
#!/bin/bash

add()
{
    result=$(( $1 + $2 ))
    return $result
}

a=3
b=5
add $a $b
sum=$?

echo "$a + $b = $sum"
```

**8. Backup script code**

```
#!/bin/bash
if [ -z $1 ] || [ -z $2 ]; then
    echo usage: $0 source_dir target_dir
else
    SRC_DIR=$1
    DST_DIR=$2
    OF=output.$(date +%y%m%d%H%M%S).tar.gz
    if [ -d $DST_DIR ]; then
        tar -cvzf $DST_DIR/$OF $SRC_DIR
    else
        mkdir $DST_DIR
        tar -cvzf $DST_DIR/$OF $SRC_DIR
    fi
fi
```

**9. 참고자료**

- <http://wiki.kldp.org/wiki.php/%C0%CO%BF%EB%C7%C1%B7%CE%B1%D7%B7%A5#s-1.1>