
BASH 셸 프로그래밍

BASH

■ Bourne Shell

- ◆ Stephen bourne에 의해 제작된 최초의 대중화된 Unix Shell

■ BASH (Bourne Again Shell)

- ◆ Bourne Shell 확장
- ◆ 다양한 프로그래밍 언어 능력 제공

셸 변수(Variable)

▣ 변하는 값을 저장할 수 있는 저장공간

◆ 사용자 변수(user variable)

- 사용자가 만들어 값을 할당
- 사용자 변수에 값을 할당하려면 "=" 기호를 사용하며, 사용자는 이러한 변수 명을 문자나 첫 글자가 문자인 숫자의 조합으로 만들
- 만약 할당하려는 변수가 존재하지 않으면 자동적으로 새로운 변수를 생성하지만, 만약 기존 변수가 존재하면 이 변수가 가지고 있는 이전 값을 새로운 값으로 변경

◆ 환경 변수(environment variable)

- 셸에 의해서 지정된 환경변수
- 셸이 수행되면 자동으로 할당되지만, 사용자가 변경 가능

셸 변수(Variable)

■ 사용자 변수 사용 예

```
libero@seps1:~ <2>
[libero@seps1 libero]$ name=libero
[libero@seps1 libero]$ echo my name is ${name}
my name is libero
[libero@seps1 libero]$
```

```
libero@seps1:~
[libero@seps1 libero]$ myname=seong-woo kim # 공백을 포함한 문자열
-bash: kim: command not found
[libero@seps1 libero]$ echo $myname

[libero@seps1 libero]$ myname="seong-woo kim" # 공백을 포함한 문자열
[libero@seps1 libero]$ echo $myname
seong-woo kim
[libero@seps1 libero]$ echo $my name is kim # "$" 를 변수로 인식
name is kim
[libero@seps1 libero]$ echo \$my name is kim # "$" 를 출력
$my name is kim
[libero@seps1 libero]$
```

```
libero@seps1:~/test
[libero@seps1 test]$ echo $list
test.c test1 test2 text.c text1.c text2.c
[libero@seps1 test]$ list=
[libero@seps1 test]$ echo $list

[libero@seps1 test]$ unset list
[libero@seps1 test]$
```

셸 변수(Variable)

■ 사용자 변수 사용 예

```
libero@seps1:~$ myname="seong-woo kim" # 다수의 공백 포함
libero@seps1:~$ echo $myname          # 하나의 공백으로 인식
seong-woo kim
libero@seps1:~$ echo "$myname"        # 다수의 공백으로 인식
seong-woo      kim
libero@seps1:~$
```

```
libero@seps1:~/test$ ls
5      for.sh      integer.sh  special.sh  test1      text1.c
abc    hello.txt  reverse.sh  special4741.sh test2      text2.c
case.sh if.sh      sample.sh  test.c      text.c     zzz
libero@seps1:~/test$ list=t*          # 메타문자로 인식
libero@seps1:~/test$ echo $list
test.c test1 test2 text.c text1.c text2.c
libero@seps1:~/test$ list="t*"
libero@seps1:~/test$ echo $list
test.c test1 test2 text.c text1.c text2.c
libero@seps1:~/test$ echo "$list"    # 특수문자 "*" 표현
t*
libero@seps1:~/test$
```

사용자 변수

■ 변수 접근 방법

정의	의미
<code>\$var</code>	var를 출력
<code>\${var}</code>	변수이름 일부로 번역
<code>\${var-word}</code>	만약 설정되어 있으면 var값을, 그렇지 않으면 word값으로 대치
<code>\${var+word}</code>	var가 설정되었을때만 word로 대치
<code>\${var=word}</code>	만약 설정되어 있지 않으면 var에 word를 할당, var 값으로 대치
<code>\${var?word}</code>	var가 설정되면 var로 대치, var가 설정되지 않으면 word는 표준에러로 출력되고 셸은 종료, 만일 word가 생략되면 그 대신 표준에러 메시지가 표시됨

사용자 변수

■ 변수 접근 방법 사용 예

```
libero@seps1:~ <2>  
[libero@seps1 libero]$ name=libero  
[libero@seps1 libero]$ echo my name is ${name}  
my name is libero  
[libero@seps1 libero]$
```

```
libero@seps1:~/test  
[libero@seps1 test]$ date=${date-'date'}  
[libero@seps1 test]$ echo $date  
일 8월 4 09:22:11 KST 2002  
[libero@seps1 test]$
```

사용자 변수

■ 변수 접근 방법 사용 예

```
libero@seps1:~  
[libero@seps1 libero]$ echo x=${var=10}  
x=10  
[libero@seps1 libero]$ echo $var  
10  
[libero@seps1 libero]$
```

```
libero@seps1:~  
[libero@seps1 libero]$ flg=0  
[libero@seps1 libero]$ echo ${flg+"flg is set"}  
flg is set  
[libero@seps1 libero]$ echo ${flg1+"flg is unset"}  
[libero@seps1 libero]$
```

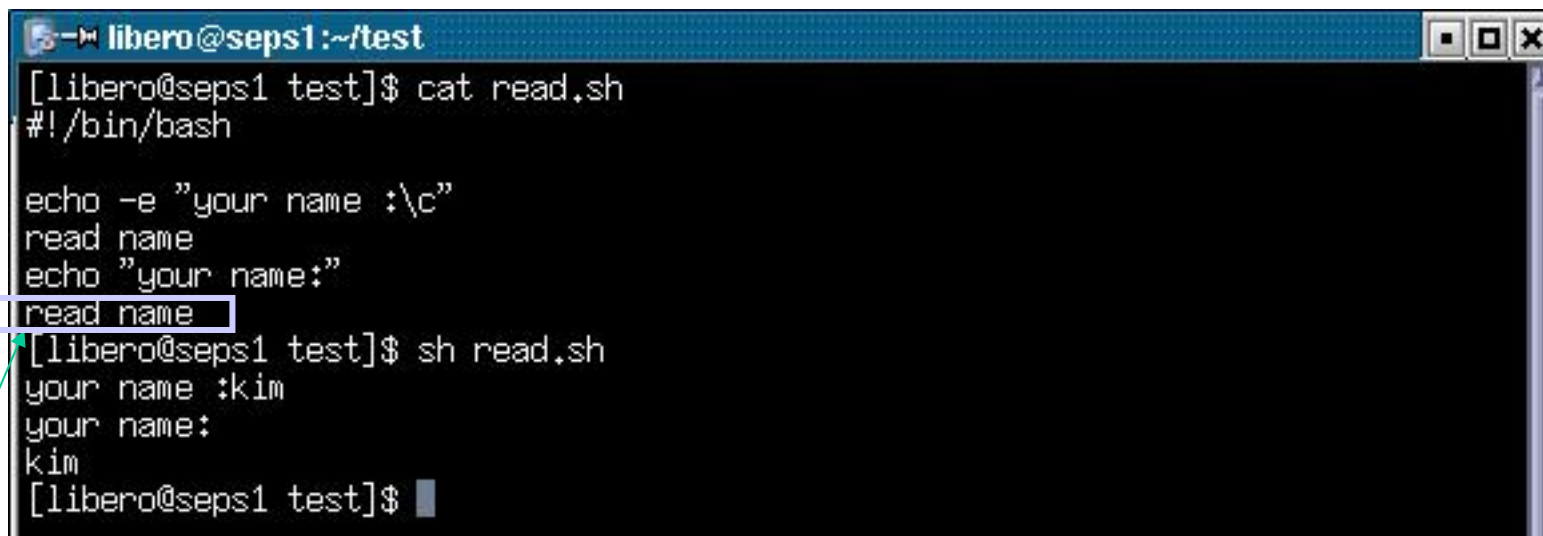
```
libero@seps1:~  
[libero@seps1 libero]$ i=10  
[libero@seps1 libero]$ val=${i?"i is not set"}  
[libero@seps1 libero]$ echo $val  
10  
[libero@seps1 libero]$ val=${j?"j is not set"} # 정의되지 않은 변수에 접근  
-bash: j: j is not set  
[libero@seps1 libero]$
```


사용자 변수

▣ “read” 명령

◆ 사용자가 표준 입력으로 입력하여 변수에 정보 저장 가능

◆ read 사용 예



```
libero@seps1:~/test
[libero@seps1 test]$ cat read.sh
#!/bin/bash

echo -e "your name :\c"
read name
echo "your name:"
read name
[libero@seps1 test]$ sh read.sh
your name :kim
your name:
kim
[libero@seps1 test]$
```

“echo \$name” 으로 수정

사용자 변수

■ 변수 사용 범위

- ◆ 사용자 변수는 해당 셸에서만 접근 가능
- ◆ 다른 셸에서도 사용하기 위해서는 “**export**” 명령 사용
- ◆ 변수 사용 범위 예

```
libero@seps1:~/test
[libero@seps1 test]$ myname=kim
[libero@seps1 test]$ echo $myname
kim
[libero@seps1 test]$ sh # 서브셸 실행
sh-2.05b$ echo $myname # 선언되지 않은 변수로 인식

sh-2.05b$ exit
exit
[libero@seps1 test]$ echo $myname
kim
[libero@seps1 test]$
```

```
libero@seps1:~/test
[libero@seps1 test]$ myname=kim
[libero@seps1 test]$ export myname
[libero@seps1 test]$ sh # 서브셸 실행
sh-2.05b$ echo $myname
kim
sh-2.05b$
```

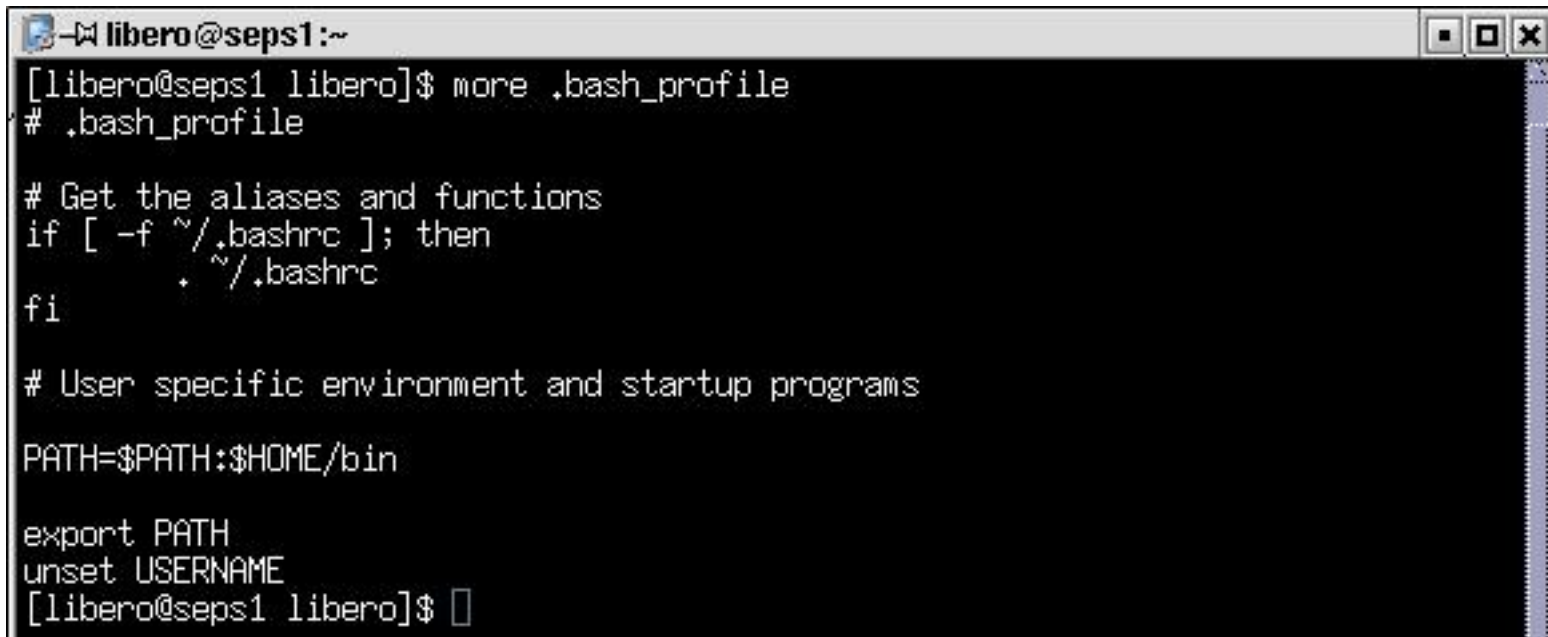
사용자 변수

▣ 변수 사용 범위 예

셸 프로그래밍

■ 주석(Comment)

- ◆ 주석을 넣기 위하여 특수한 기호 "#"를 사용
- ◆ "#"기호를 라인의 맨 처음에 두면 그 라인 전체가 주석으로 처리
- ◆ 명령문 뒤에 "#" 기호를 넣으면 그 기호 뒤부터 주석으로 인식
- ◆ 주석 사용 예



```
libero@seps1:~  
[libero@seps1 libero]$ more .bash_profile  
# .bash_profile  
  
# Get the aliases and functions  
if [ -f ~/.bashrc ]; then  
    . ~/.bashrc  
fi  
  
# User specific environment and startup programs  
  
PATH=$PATH:$HOME/bin  
  
export PATH  
unset USERNAME  
[libero@seps1 libero]$
```

셸 프로그래밍

■ Here 자료

◆ 셸 스크립트 내에 포함된 데이터(**here data**) 를 표준입력으로 사용 가능

➤ <참고> 보통의 셸 스크립트에서의 입력 데이터는 리다이렉션을 이용하거나 별도의 파일을 이용

◆ "here" 자료는 "<<"로 시작하고, "<<" 뒤에 나오는 문자열이 다시 나오는 부분까지 범위 지정

◆ 형식

<< 구분자

Here 자료

구분자

셸 프로그래밍

■ Here 자료 예제

```
libero@seps1:~/test
[libero@seps1 test]$ cat test.txt
kim 0.1 30 korea
kang 80 20 korea
min 40 50 china
sim 59 80 france
[libero@seps1 test]$ grep kim test.txt      # grep 명령을 이용하여 "kim" 검색
kim 0.1 30 korea
[libero@seps1 test]$ cat test1             # here 자료를 이용하여 "kim" 검색
#!/bin/sh
grep -i "$1" << EOF      # here 자료 시작
kim 0.1 30 korea
kang 80 20 korea
min 40 50 china
sim 59 80 france
EOF                          # here 자료 끝
[libero@seps1 test]$ ./test1 kim
kim 0.1 30 korea
[libero@seps1 test]$
```

셸 프로그래밍

■ 종료상태(Exit Status)

- ◆ 셸 명령어들은 실행이 끝난 후 실행 상태를 반환
 - 정상 종료일 경우 0, 아니면 실패하면 0 이 아닌 값을 반환
- ◆ "\$?"은 가장 최근에 실행된 명령의 반환값(성공여부)를 가짐
- ◆ 종료상태 사용 예

```
libero@seps1:~/test
[libero@seps1 test]$ echo exit # 정상 종료
exit
[libero@seps1 test]$ echo $?
0
[libero@seps1 test]$ eecho exit # 비정상 종료
-bash: eecho: command not found
[libero@seps1 test]$ echo $?
127
[libero@seps1 test]$
```

셸 프로그래밍

■ 종료상태(Exit Status)

◆ 종료상태 사용 예 2

```
libero@seps1:~/test
[libero@seps1 test]$ cat params.sh
#!/bin/bash

echo "total1 params=$*"
echo "total2 params number=$#"
exit 10 # 종료값 10 으로 설정
echo "total3 params number=$#"
[libero@seps1 test]$ echo $? # 정상 종료
0
[libero@seps1 test]$ ./params.sh 1 2 3 # 셸 수행중 정지
total1 params=1 2 3
total2 params number=3
[libero@seps1 test]$ echo $? # 임의의 종료 조건값 지정
10
[libero@seps1 test]$
```


셸 프로그래밍

연산식과 문자열 연산자

◆ Bourne 셸에서는 자체 산술 연산을 지원하지 않음

➤ “expr” 명령 사용

◆ 연산식

연산자	의미
*, / , %	곱셈, 나눗셈, 나머지 연산
+, -	덧셈, 뺄셈
=, >, >=, <, <=, !=	비교연산자
&	논리적 논리곱(AND)
	논리적 논리합(OR)

◆ 문자열 연산자

연산자	의미
match	일치하는 문자열 길이 반환
substr	부문자열 추출
index	부문자열 위치 반환
length	문자열 길이를 반환

셸 프로그래밍

■ expr 사용 예

```
libero@seps1:~/test
[libero@seps1 test]$ a=10
[libero@seps1 test]$ b='expr $a+10' # 연산자 사이에 공백을 두지 않으면 문자열로 인식
[libero@seps1 test]$ echo $b
10+10
[libero@seps1 test]$ b='expr $a + 10'
[libero@seps1 test]$ echo $b
20
[libero@seps1 test]$ c='expr $b + 10 / 2'
[libero@seps1 test]$ echo $c
25
[libero@seps1 test]$ d='expr \( 5 + 10 \) \ / 2' # 메타문자를 사용하려면 "\" 사용
[libero@seps1 test]$ echo $d
7
[libero@seps1 test]$ d='expr \( $a + 10 \) \ / 2'
[libero@seps1 test]$ echo $d
10
[libero@seps1 test]$ echo 'expr 4 \ > 5' # 거짓일 때 "0"을 반환
0
[libero@seps1 test]$ echo 'expr \( 4 \ > 5 \) \ | \( 7 \ > 4 \)' # 둘 중 하나가 참
1
[libero@seps1 test]$ █

[한글][완성][두벌식]
```

셸 프로그래밍

□ 조건식

◆ “test” 명령 사용

옵션	의미	기능	옵션	의미	기능
()	연산의 순서를 제어, 그룹		-z string	string 의 길이가 “0” 이면 참	문자열 연산
!	부정(NOT)	논리 연산자	-n string	string 의 길이가 “0” 아니면 참	
-a	논리곱(AND)		string1 = string2	string1 과 string2 가 같으면 참	
-o	논리합(OR)		string1 != string2	string1 과 string2 가 다르면 참	
n1 -eq n2	두 수가 같으면 참	숫자 연산자	-r file	file이 존재하고 읽을수 있으면 참	파일 연산자
n1 -ne n2	두 수가 같지 않으면 참		-w file	file이 존재하고 기록 가능하면 참	
n1 -gt n2	n1 이 크면 참		-x file	file이 존재하고 실행 가능하면 참	
n1 -ge n2	n1 이 크거나 같으면 참		-f file	file이 존재하고 정규 파일이면 참	
n1 -lt n2	N1 이 작으면 참		-d file	file이 존재하고 디렉토리이면 참	
n1 -le n2	N1 이 작거나 같으면 참		-s file	file이 존재하고 비지 않으면 참	

셸 프로그래밍

■ test 사용 예

```
libero@seps1:~ <2>
[libero@seps1 libero]$ test 4 -eq 5
[libero@seps1 libero]$ echo $?
1
[libero@seps1 libero]$ test 5 -eq 5
[libero@seps1 libero]$ echo $?
0
[libero@seps1 libero]$ a=kim
[libero@seps1 libero]$ b=kang
[libero@seps1 libero]$ test $a = "kim" # 변수 비교
[libero@seps1 libero]$ echo $?
0
[libero@seps1 libero]$ test \( $a = "kim" \) -a \( $b = "kang" \) # 논리 연산자 이용
[libero@seps1 libero]$ echo $?
0
[libero@seps1 libero]$
```

제어 구조

■ 조건문(if ~ else, case)

- ◆ 조건문은 조건이 만족할 때에만 문장을 실행시킴

■ “if” 문

- ◆ 조건이 만족하면 “then” 이하의 문장 실행, “fi” 가 나오면 종료

◆ 형식

➤ “if ~ then ~ [elif ~ then ~] [else ~] fi”

참고) ~ : 리스트(Lists)의 종료 상태값을 따름

- ;, &, <개행문자> 중 하나로 끝나는 연속된 문자열
- 연산자는 ;(연속실행), &(백그라운드), &&(AND 제어연산자), || (OR 제어연산자) 가능

■ "case" 문

- ◆ "if"문의 확장으로 다중 선택 형식

- ◆ 몇 개의 패턴에서 해당하는 것을 찾아 그 다음의 명령들을 실행

◆ 형식

➤ “case 단어 in [패턴 [| 패턴]) ~ ;;] ~ ;; esac”

제어 구조

■ if 문 사용 예 (파일 존재 여부 출력 스크립트)

```
libero@seps1:~/test
[libero@seps1 test]$ cat if.sh
#!/bin/sh

if [ -f $1 ]
then
    echo "file exists!"
fi
[libero@seps1 test]$ ./if.sh test.c # 파일이 존재
file exists!
[libero@seps1 test]$ echo $?
0
[libero@seps1 test]$ ./if.sh a.c    # 파일이 존재하지 않음
[libero@seps1 test]$ echo $?
0
[libero@seps1 test]$
```

제어 구조

■ if 문 사용 예 (정수 구분 스크립트)

```
libero@seps1:~/test
[libero@seps1 test]$ cat integer.sh
#!/bin/bash

echo -e "Enter number: \c"
read num
if [ $num -gt 0 ]
then
    echo "Positive number!";
elif [ $num -eq 0 ]
then
    echo "Zero number!";
else
    echo "Negative number!";
fi
[libero@seps1 test]$ ./integer.sh
Enter number: 0
Zero number!
[libero@seps1 test]$ ./integer.sh
Enter number: 9
Positive number!
[libero@seps1 test]$ ./integer.sh
Enter number: -1
Negative number!
[libero@seps1 test]$
```

제어 구조

■ case 문 사용 예 (패턴에 따른 명령 수행 스크립트)

```
libero@seps1:~/test
[libero@seps1 test]$ cat case.sh
#!/bin/sh

echo -e "Enter number: \c"
read num
case $num in
  "1")
    date
    ;;
  "a"|"A"| [A-C])
    pwd
    ;;
  "cd"|"CD")
    echo $HOME
    ;;
  *)
    echo "Usage : Input number?"
    ;;
esac
[libero@seps1 test]$ ./case.sh           # 숫자 패턴 일치
Enter number: 1
목  4월 10 11:37:29 KST 2003
[libero@seps1 test]$ ./case.sh         # 문자 패턴 일치
Enter number: a
/home/libero/test
[libero@seps1 test]$ ./case.sh        # 문자열 패턴 일치
Enter number: cd
/home/libero
[libero@seps1 test]$ ./case.sh        # 패턴이 일치하지 않을 때
Enter number: ooo
Usage : Input number?
[libero@seps1 test]$
```


제어 구조

■ 반복문(For, While, Until)

◆ 조건식이 만족하거나 만족하지 않을 때까지 반복해서 실행

■ “For” 문

◆ 단어 리스트 안의 각 멤버에 대하여 명령 리스트를 한번씩 실행

◆ 형식

➤ “for 이름 [in 단어;] do ~ done”

■ “While” 문

◆ 조건식이 참일 때에만 주어진 명령을 실행

◆ 형식

➤ “while ~ do ~ done”

■ “Until” 문

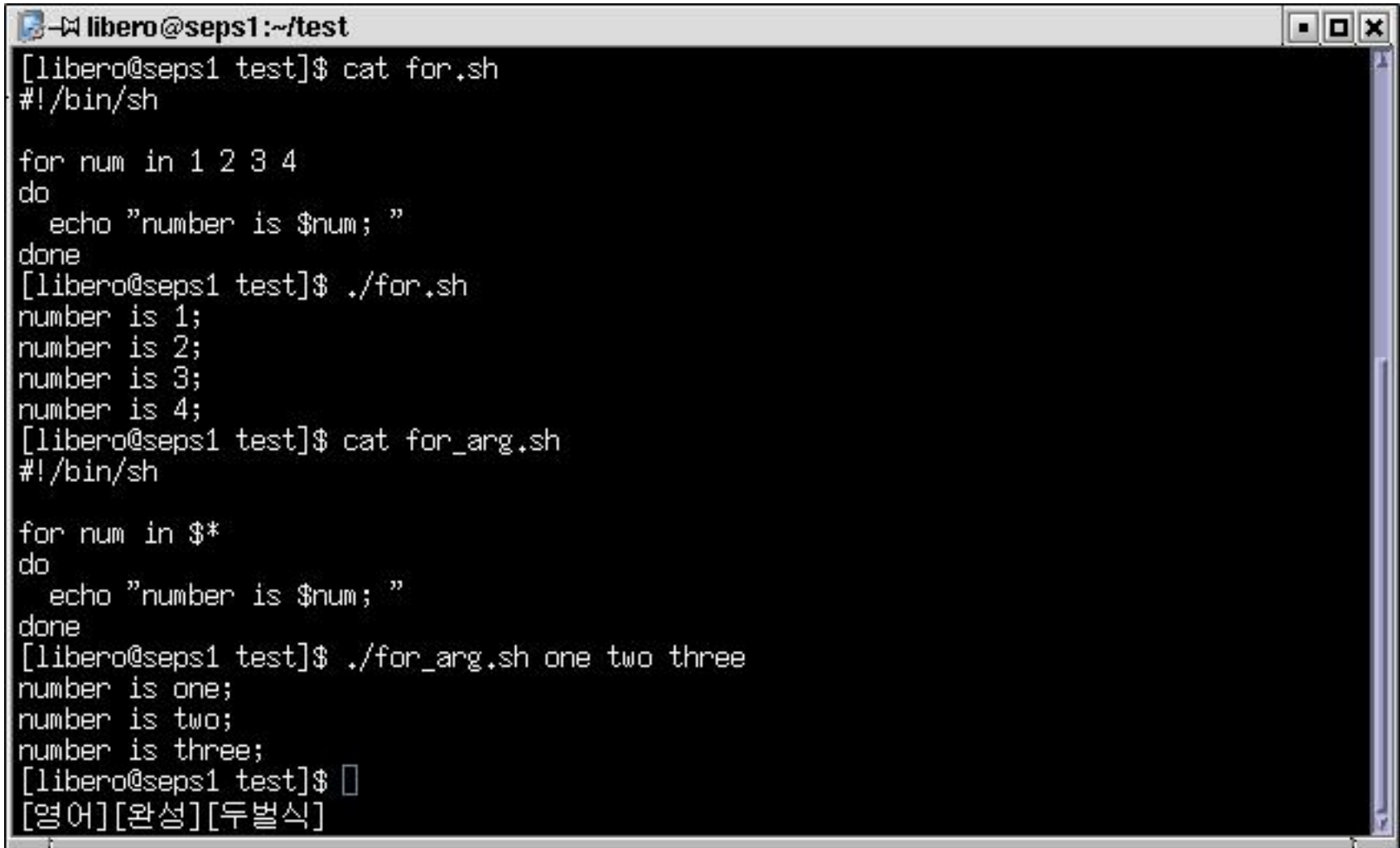
◆ “while” 문과 비슷하나, 조건식이 만족할 때까지 반복 해서 실행

◆ 형식

➤ “until ~ do ~ done”

제어 구조

■ for 문 사용 예

A terminal window titled 'libero@seps1:~/test' showing the execution of two shell scripts. The first script, 'for.sh', uses a for loop to iterate over the numbers 1, 2, 3, and 4, printing 'number is \$num;'. The second script, 'for_arg.sh', uses a for loop to iterate over the arguments 'one', 'two', and 'three', also printing 'number is \$num;'.

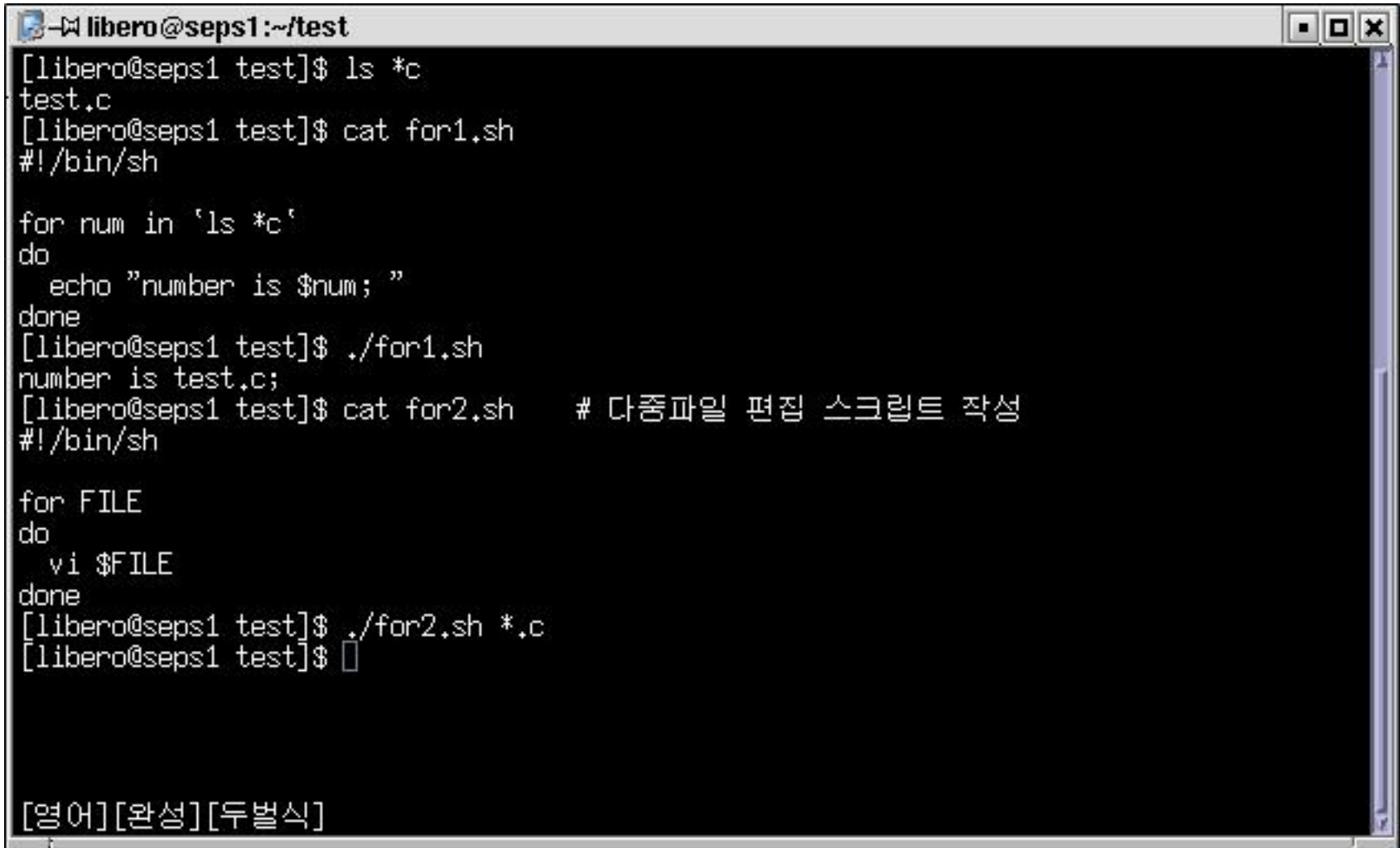
```
libero@seps1:~/test
[libero@seps1 test]$ cat for.sh
#!/bin/sh

for num in 1 2 3 4
do
    echo "number is $num;"
done
[libero@seps1 test]$ ./for.sh
number is 1;
number is 2;
number is 3;
number is 4;
[libero@seps1 test]$ cat for_arg.sh
#!/bin/sh

for num in $*
do
    echo "number is $num;"
done
[libero@seps1 test]$ ./for_arg.sh one two three
number is one;
number is two;
number is three;
[libero@seps1 test]$
```

제어 구조

■ for 문 사용 예

A terminal window titled 'libero@seps1:~/test' showing a series of shell commands and their outputs. The user lists files with 'ls *c', creating a file 'test.c'. They then create a script 'for1.sh' with a for loop that iterates over 'ls *c' and prints 'number is \$num;'. Running './for1.sh' produces the output 'number is test.c;'. Next, they create 'for2.sh' with a for loop that iterates over 'FILE' and runs 'vi \$FILE'. Finally, they run './for2.sh *.c' and the prompt returns. The window title bar shows standard window controls (minimize, maximize, close).

```
libero@seps1:~/test
[libero@seps1 test]$ ls *c
test.c
[libero@seps1 test]$ cat for1.sh
#!/bin/sh

for num in `ls *c`
do
    echo "number is $num;"
done
[libero@seps1 test]$ ./for1.sh
number is test.c;
[libero@seps1 test]$ cat for2.sh    # 다중파일 편집 스크립트 작성
#!/bin/sh

for FILE
do
    vi $FILE
done
[libero@seps1 test]$ ./for2.sh *.c
[libero@seps1 test]$
```

[영어][완성][두벌식]

제어 구조

■ while, until 문 사용 예

```
libero@seps1:~/test
[libero@seps1 test]$ cat sum.sh
#!/bin/sh

i=0          # 변수 초기화
total=0
while [ $i -le 10 ]    # 10 일 때까지만 반복
do
    total='expr $total + $i'
    i='expr $i + 1'    # 1 씩 증가
done
echo "total = $total"
[libero@seps1 test]$ ./sum.sh
total = 55
[libero@seps1 test]$ cat sum1.sh
#!/bin/sh

i=0          # 변수 초기화
total=0
until [ $i -gt 10 ]   # 변수가 10보다 커지면 반복 종료
do
    total='expr $total + $i'
    i='expr $i + 1'    # 1 씩 증가
done
echo "total = $total"
[libero@seps1 test]$ ./sum1.sh
total = 55
[libero@seps1 test]$ 
[영어][완성][두벌식]
```

제어 구조

■ Until 문 사용 예

```
libero@seps1:~/test
[libero@seps1 test]$ cat rm.sh
#!/bin/sh

input=""
until [ "$input" = "y" -o "$input" = "n" ]      # 사용자 응답 확인
do
    echo "delete subdir : $1"
    echo -e "Are you sure? \c"
    read input
done
if [ $input = "y" ]                             # "y" 나 "n" 이면 명령 실행
then
    rm -r $1
    echo "$1 is deleted!!"
else
    echo "$1 is not deleted!!"
fi
[libero@seps1 test]$ ./rm.sh imsi
delete subdir : imsi
Are you sure? d
delete subdir : imsi
Are you sure? n
imsi is not deleted!!
[libero@seps1 test]$ ./rm.sh imsi
delete subdir : imsi
Are you sure? y
rm: cannot lstat 'imsi': 그런 파일이나 디렉토리가 없음
imsi is deleted!!
[libero@seps1 test]$ █
[영어][완성][두벌식]
```

제어 구조

■ 분기명령(Break, Continue)

◆ 반복문을 빠져 나오거나 끝으로 이동하기 위한 명령

■ "break" 명령

◆ 반복에서 완전히 빠져 나옴

◆ 반복문에서 나온 후 "done"절 뒤의 문장으로 실행 계속

■ "Continue" 명령

◆ 반복의 처음부분으로, 즉 조건문으로 실행을 옮김

제어 구조

■ Continue 명령 사용 예

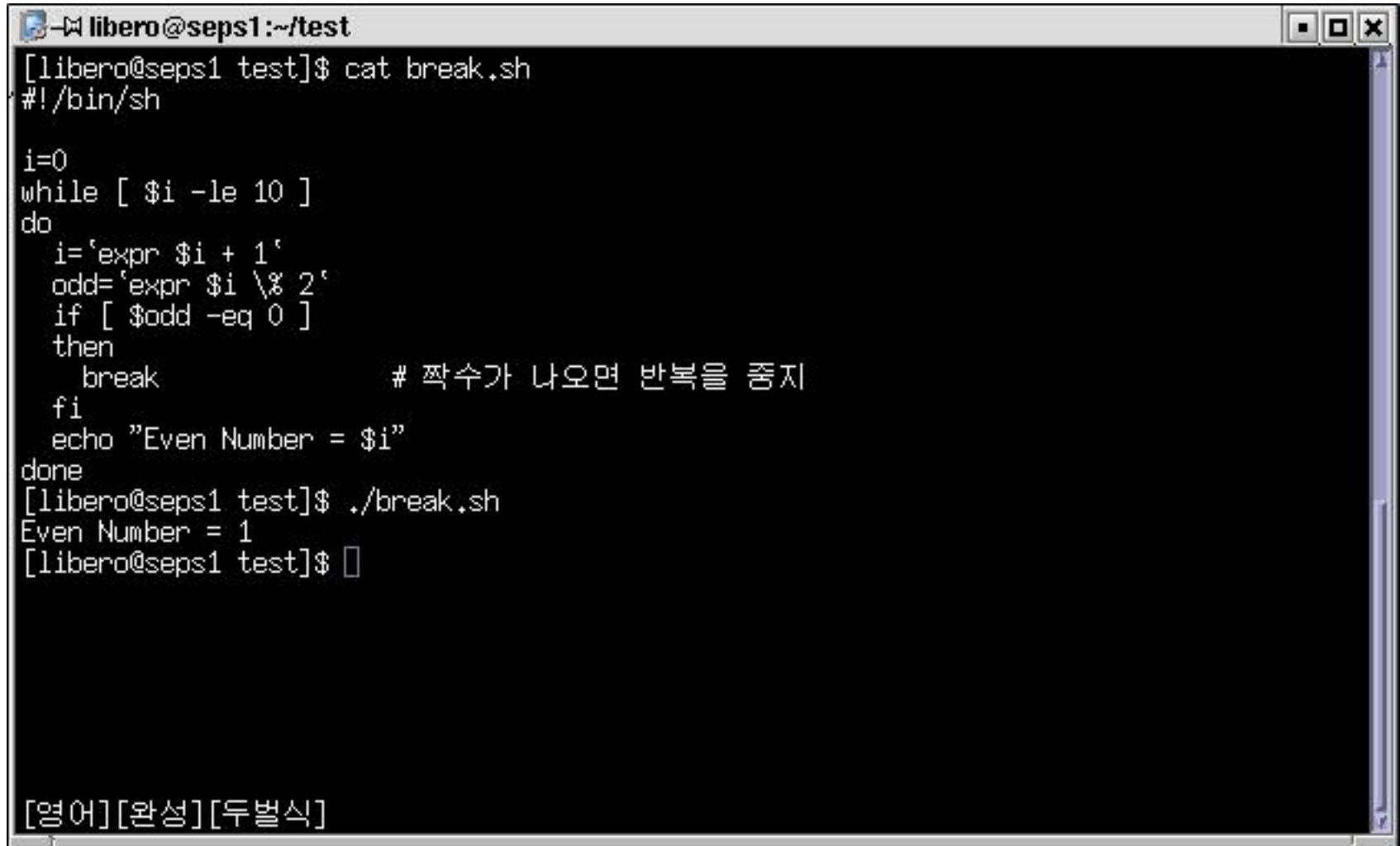
```
libero@seps1:~/test
[libero@seps1 test]$ cat even.sh
#!/bin/sh

i=0
while [ $i -le 10 ]
do
    i=`expr $i + 1`
    odd=`expr $i \% 2`      # 나머지 연산자를 이용하여 홀수, 짝수 구분
    if [ $odd -eq 1 ]
    then
        continue          # 홀수일 때는 다시 조건문으로 제어를 옮김
    fi
    echo "Even Number = $i"
done
[libero@seps1 test]$ ./even.sh
Even Number = 2
Even Number = 4
Even Number = 6
Even Number = 8
Even Number = 10
[libero@seps1 test]$
```

[영어][완성][두벌식]

제어 구조

■ Break 명령 사용 예



```
libero@seps1:~/test
[libero@seps1 test]$ cat break.sh
#!/bin/sh

i=0
while [ $i -le 10 ]
do
    i=`expr $i + 1`
    odd=`expr $i \% 2`
    if [ $odd -eq 0 ]
    then
        break          # 짝수가 나오면 반복을 중지
    fi
    echo "Even Number = $i"
done
[libero@seps1 test]$ ./break.sh
Even Number = 1
[libero@seps1 test]$
```

[영어][완성][두벌식]

제어 구조

■ “select” 명령

◆ 사용자가 선택할 옵션 메뉴를 제공하여 명령을 수행

◆ 형식

➤ “select 이름 [in 단어;] do ~ done”

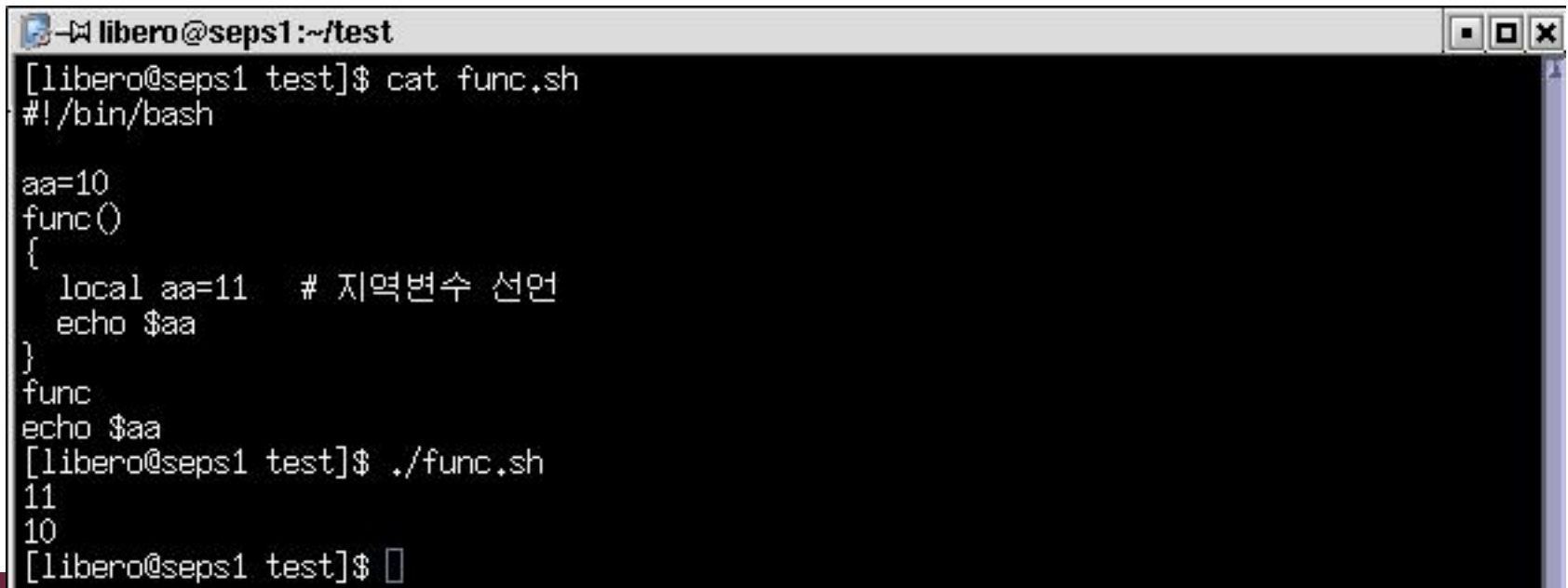
◆ 사용 예

```
libero@seps1:~/test
[libero@seps1 test]$ cat select.sh
#!/bin/bash

select file in `ls i*`
do
    echo "This file is $file"
done
[libero@seps1 test]$ ./select.sh
1) i.out
2) if.sh
3) if1.sh
4) integer.sh
#? 1
This file is i.out
#? █
```

셸 함수

- BASH 는 함수 기능을 제공
- 장점
 - ◆ 실행속도가 빠르고, 모듈화 제공
- 형식
 - ◆ [function] 이름 () { ~; }
- 사용 예

A terminal window titled 'libero@seps1:~/test' showing the execution of a shell script. The script defines a function 'func' that uses a local variable 'aa' and prints its value. The function is then called, demonstrating that the local variable's value is preserved within the function's scope.

```
libero@seps1:~/test
[libero@seps1 test]$ cat func.sh
#!/bin/bash

aa=10
func()
{
    local aa=11    # 지역변수 선언
    echo $aa
}
func
echo $aa
[libero@seps1 test]$ ./func.sh
11
10
[libero@seps1 test]$
```

셸 내부 명령

■ set 명령

- ◆ 셸 스크립트를 작성한 후 이 스크립트가 이상이 있는지 아니면 어디서 에러가 발생했는지를 알아보기 위해 사용

옵션	의미
e	명령이 실패하면 트랩이 실행되고 종료
n	명령을 실행하지 않고 받아들임
t	다음 명령을 실행하고 종료
u	정의되지 않은 변수를 만날 때 에러 발생
v	셸 스크립트를 그대로 보여줌
x	실행한 결과를 보여줌

셸 내부 명령

■ set 사용 예

```
libero@seps1:~/test
[libero@seps1 test]$ cat set.sh
#!/bin/sh

set -xv                # 디버깅 옵션 지정
i=0
while [ $i -le 2 ]
do
    i=`expr $i + 1`
    echo "Even Number = $i"
done
[libero@seps1 test]$ ./set.sh
i=0
+ i=0
while [ $i -le 2 ]
do
    i=`expr $i + 1`
    echo "Even Number = $i"
done
+ '[' 0 -le 2 ']'
expr $i + 1
++ expr 0 + 1
+ i=1
+ echo 'Even Number = 1'
Even Number = 1
+ '[' 1 -le 2 ']'
expr $i + 1
++ expr 1 + 1
+ i=2
+ echo 'Even Number = 2'
Even Number = 2
+ '[' 2 -le 2 ']'
expr $i + 1
++ expr 2 + 1
+ i=3
+ echo 'Even Number = 3'
Even Number = 3
+ '[' 3 -le 2 ']'
[libero@seps1 test]$ █
[영어][완성][두벌식]
```