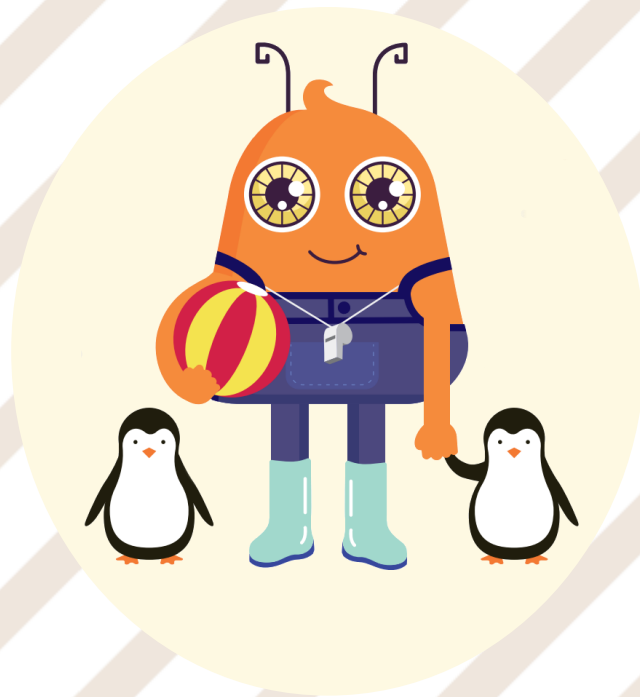


11

CHAPTER

셸 스크립트 프로그래밍



C.ontents

- 01** 셀의 개념과 특징
- 02** 셀 스크립트 프로그래밍 기본
- 03** 셀 스크립트 프로그래밍 응용

학습목표

- 셀의 개념과 명령문 처리 방식을 이해한다.
- 셀 스크립트의 작성과 실행 방법을 익힌다.
- 다양한 문법을 활용하여 셀 스크립트를 작성한다.

1-1 셸의 개념

- 리눅스의 셸
 - 명령과 프로그램을 실행할 때 사용하는 인터페이스
 - 사용자가 입력한 명령을 해석하여 커널에 전달하거나 커널의 처리 결과를 사용자에게 전달하는 역할
 - 셸은 Server(B)의 텍스트 모드처럼 명령을 입력하는 환경과 비슷
 - 우분투 에서 기본적으로 사용하는 셸은 bash(Bourne Again Shell)이며 배시셸이라고 읽음
- bash의 특징
 - 에일리어스(alias, 명령 단축) 기능
 - 히스토리 기능(↑ 또는 ↓)
 - 연산 기능
 - Job Control 기능
 - 자동 이름 완성 기능(Tab)
 - 프롬프트 제어 기능
 - 명령 편집 기능

1-2 셸의 명령문 처리 방법

- 셸 명령문의 형식
 - (프롬프트) 명령 [옵션...] [인자...]
- 셸 명령문의 예

```
# ls -l
# rm -rf /mydir
# find . / -name "*.conf"
```

1-3 환경 변수

■ 실행 방법

- 셀에서는 여러 가지 환경 변수 값을 불러올 수 있음
- 설정된 환경 변수는 **echo \$환경변수** 명령으로 확인
- 호스트 이름을 출력하려면 **echo \$HOSTNAME** 명령을 실행
- **export 환경변수=값** 명령을 실행하면 환경 변수 값을 변경(확인할 때는 **printenv** 명령 실행)

표 11-1 배시셸의 주요 환경 변수

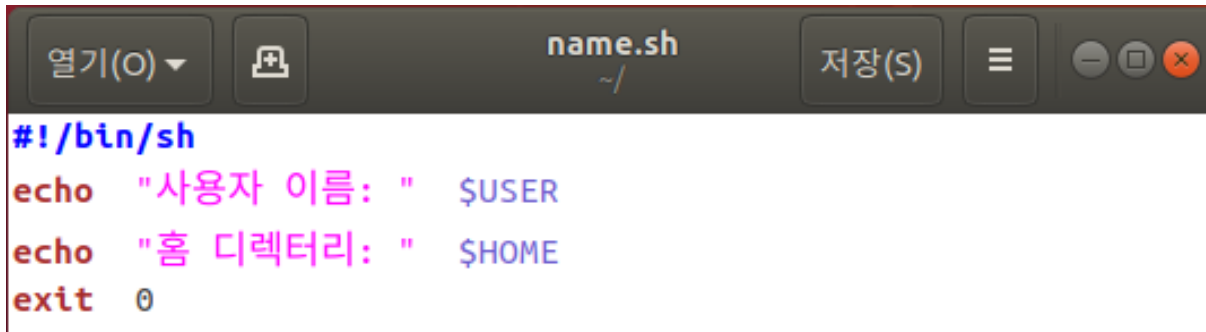
| 환경 변수 | 설명 | 환경 변수 | 설명 |
|----------|----------------|--------------|----------------------|
| HOME | 현재 사용자의 홈 디렉터리 | PATH | 실행 파일을 찾는 디렉터리 경로 |
| LANG | 기본 자원 언어 | PWD | 사용자의 현재 작업 디렉터리 |
| TERM | 로그인 터미널 유형 | SHELL | 로그인해서 사용하는 셸 |
| USER | 현재 사용자의 이름 | DISPLAY | X 디스플레이 이름 |
| COLUMNS | 현재 터미널의 칼럼 수 | LINES | 현재 터미널의 라인 수 |
| PS1 | 1차 명령 프롬프트 변수 | PS2 | 2차 명령 프롬프트(대개 >) |
| BASH | 배시셸의 경로 | BASH_VERSION | 배시셸의 버전 |
| HISTFILE | 히스토리 파일의 경로 | HISTSIZE | 히스토리 파일에 저장되는 명령어 개수 |
| HOSTNAME | 호스트 이름 | USERNAME | 현재 사용자의 이름 |
| LOGNAME | 로그인 이름 | LS_COLORS | ls 명령의 확장자 색상 옵션 |
| MAIL | 메일을 보관하는 경로 | OSTYPE | 운영체제 유형 |

2-1 셸 스크립트 작성

■ 작성 방법

- 셸 스크립트도 일반적인 프로그래밍 언어와 비슷하게 변수, 반복문, 제어문 등을 사용
- 별도로 컴파일하지 않고 텍스트 파일 형태로 셸에서 바로 실행
- 셸 스크립트는 주로 vi 에디터나 gedit로 작성

■ vi name.sh 또는 gedit name.sh 명령 실행



```
#!/bin/sh
echo "사용자 이름: " $USER
echo "홈 디렉터리: " $HOME
exit 0
```

- 1행: 첫 행에 반드시 써야 하며, 특별한 형태의 주석(#!)으로 배시셸을 사용하겠다는 의미
- 2행: echo는 화면에 출력하는 명령어. 먼저 '사용자 이름 :'이라는 글자를 출력하고 옆에 \$USER라는 환경 변수의 내용을 출력
- 4행: 종료 코드를 반환. 0은 성공을 의미

2-2 셸 스크립트 실행

- **sh** 명령으로 실행

- **sh 스크립트파일** 명령으로 실행하는 방법은 셸 스크립트 파일의 속성을 변경할 필요가 없다는 것이 장점

```
root@server: ~
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
root@server:~# ls -l name.sh
-rw-r--r-- 1 root root 86  7월  23 19:40 name.sh
root@server:~#
root@server:~# sh name.sh
사용자 이름: root
홈 디렉터리: /root
root@server:~#
```

- '실행 가능' 속성으로 변경 후 실행

- 먼저 셸 스크립트 파일의 속성을 '실행 가능'으로 변경한 후 **./스크립트파일** 명령을 실행
- **chmod +x 파일명**은 현재 파일의 속성에 '실행 가능' 속성을 추가하라는 명령

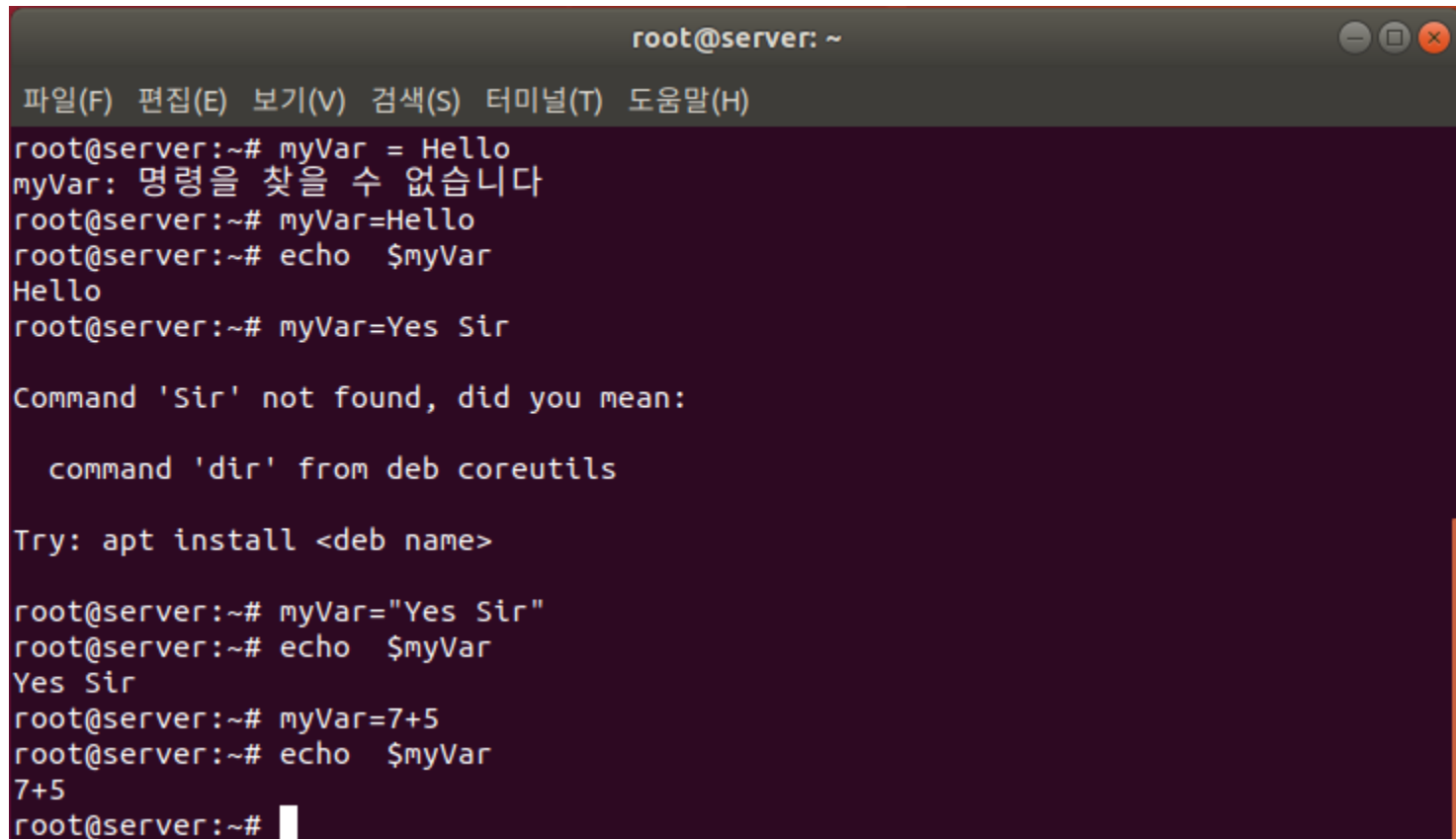
```
root@server: ~
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
root@server:~# ls -l name.sh
-rw-r--r-- 1 root root 86  7월  23 19:40 name.sh
root@server:~# chmod +x name.sh
root@server:~#
root@server:~# ls -l name.sh
-rwxr-xr-x 1 root root 86  7월  23 19:40 name.sh
root@server:~# ./name.sh
사용자 이름: root
홈 디렉터리: /root
root@server:~#
```


2-3 변수

- 변수의 개요
 - 변수는 값을 계속 변경하여 저장하는 개념
 - 셸 스크립트의 구조는 변경할 필요가 없는데 설정 값이 상황에 따라 다를 때는 변수를 바꾸는 방식으로 프로그래밍하면 편리
 - 셸 스크립트에서는 변수를 사용하기 전에 미리 선언하지 않으며, 처음 변수에 값이 할당되면 자동으로 변수가 생성
 - 변수에 들어가는 모든 값은 문자열(string)로 취급되고 숫자를 넣어도 동일하게 취급
 - 변수명은 대문자와 소문자를 구분하기 때문에 \$aa와 \$AA는 다른 변수명
 - 변수를 대입할 때 '=' 앞뒤에 공백이 없어야 함

2-3 변수

```
myVar = Hello -- 오류! = 앞뒤에 공백이 있음
myVar=Hello
myVar=Yes Sir -- 오류! 값의 공백은 " "로 묶어야 함
myVar="Yes Sir"
myVar=7+5      -- 정상이지만 '7+5'라는 문자열로 인식
```



```
root@server: ~
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
root@server:~# myVar = Hello
myVar: 명령을 찾을 수 없습니다
root@server:~# myVar=Hello
root@server:~# echo $myVar
Hello
root@server:~# myVar=Yes Sir

Command 'Sir' not found, did you mean:

  command 'dir' from deb coreutils

Try: apt install <deb name>

root@server:~# myVar="Yes Sir"
root@server:~# echo $myVar
Yes Sir
root@server:~# myVar=7+5
root@server:~# echo $myVar
7+5
root@server:~#
```

2-3 변수

■ 변수의 입력과 출력

- \$가 포함된 글자를 출력하려면 ' '로 묶거나 앞에 \를 넣어야 함
- " "로 변수를 묶거나 묶지 않아도 출력

var1.sh

```
1  #!/bin/sh
2  myvar="Hi Woo"
3  echo $myvar    -- 'Hi Woo' 출력
4  echo "$myvar"  -- 3행과 동일한 효과
5  echo '$myvar'  -- '$myvar' 출력
6  echo \myvar    -- 5행과 동일한 효과
7  echo 값 입력 :
8  read myvar     -- 변수 myvar에 키보드로 값(문자열) 입력
9  echo '$myvar' = $myvar
10 exit 0
```

```
root@server: ~
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
root@server:~# sh var1.sh
Hi Woo
Hi Woo
$myvar
$myvar
값 입력 :
쿡북 비기너입니다.
$myvar = 쿡북 비기너입니다.
root@server:~#
```

2-3 변수

■ 숫자 계산

- 변수 값을 +, -, *, / 등으로 연산하려면 `expr` 키워드 사용
- 단, 수식과 함께 키보드의 1 왼쪽에 있는 백쿼트(````)로 반드시 묶어야 함
- 수식에 괄호를 사용하려면 그 앞에 반드시 `w(w)`를 넣어야 함
- +, -, /와 달리 *도 예외적으로 앞에 `w(w)`를 넣어야 함

numcalc.sh

```
1  #!/bin/sh
2  num1=100
3  num2=$num1+200
4  echo $num2
5  num3=`expr $num1 + 200`
6  echo $num3
7  num4=`expr \( $num1 + 200 \) / 10 \* 2`
8  echo $num4
9  exit 0
```

2-3 변수

```
root@server: ~
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
root@server:~# sh numcalc.sh
100+200
300
60
root@server:~#
```

- 3행: 문자열로 취급하며 모두 붙여서 써야 함
- 5행: 숫자로 취급하여 계산하며 각 단어를 띄어쓰기해야 함
- 7행: 괄호와 * 앞에는 w(w)를 넣음

2-3 변수

■ 파라미터 변수

- 파라미터 변수는 \$0, \$1, \$2... 형태(실행하는 명령의 각 부분을 변수로 지정한다는 의미)
- **apt-get -y install gftp** 명령의 경우 파라미터 변수를 지정
- \$0에는 apt-get이, \$1에는 -y가, \$2에는 install이, \$3에는 gftp가 저장
- 명령 전체의 파라미터 변수는 \$*로 나타냄

표 11-2 파라미터 변수 지정의 예

| 명령 | apt-get | -y | install | gftp |
|---------|---------|-----|---------|------|
| 파라미터 변수 | \$0 | \$1 | \$2 | \$3 |

paravar.sh

```
1 #!/bin/sh
2 echo "실행파일 이름은 <$0>이다"
3 echo "첫번째 파라미터는 <$1>이고, 두번째 파라미터는 <$2>다"
4 echo "전체 파라미터는 <$*>다"
5 exit 0
```

```
root@server: ~
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
root@server:~# sh paravar.sh 1번 2번 3번
실행파일 이름은 <paravar.sh>이다
첫번째 파라미터는 <1번>이고, 두번째 파라미터는 <2번>다
전체 파라미터는 <1번 2번 3번>다
root@server:~#
```

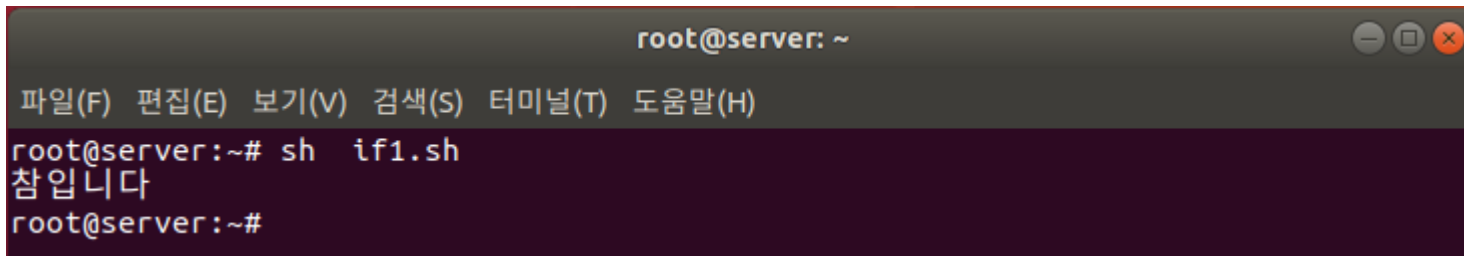
2-4 if문과 case문

- 기본 if문

- [조건] 안의 각 단어 사이에 공백이 있어야 한다는 것을 주의

```
if [ 조건 ]  
then  
    참인 경우 실행  
fi
```

```
if1.sh  
  
1  #!/bin/sh  
2  if [ "cook" = "cook" ]  
3  then  
4      echo "참입니다"  
5  fi  
6  exit 0
```



A terminal window titled 'root@server: ~' with standard window controls. The menu bar shows '파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)'. The command prompt is 'root@server:~#'. The user enters 'sh if1.sh', and the terminal outputs '참입니다' followed by a new prompt 'root@server:~#'.

```
root@server: ~  
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)  
root@server:~# sh if1.sh  
참입니다  
root@server:~#
```

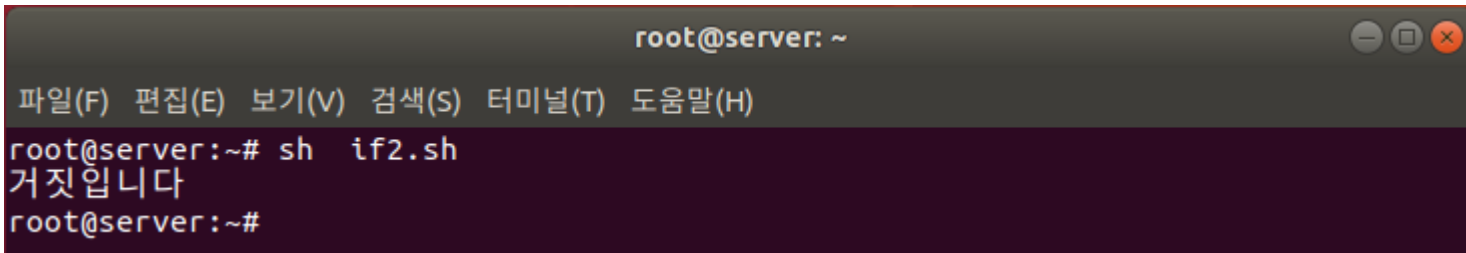
- 2행의 [] 안에는 참과 거짓을 구분하는 조건식이 들어감
- '='는 문자열이 같은지 비교하고 '!='는 문자열이 같지 않은지 비교
- if1.sh에서는 조건식이 참이므로 4행을 실행

2-4 if문과 case문

- if~else문
 - 참인 경우와 거짓인 경우를 구분하여 실행

```
if [ 조건 ]  
then  
    참인 경우 실행  
else  
    거짓인 경우 실행  
fi
```

```
if2.sh  
  
1  #!/bin/sh  
2  if [ "cook" != "cook" ]  
3  then  
4      echo "참입니다"  
5  else  
6      echo "거짓입니다"  
7  fi  
8  exit 0
```



A terminal window titled 'root@server: ~' with standard window controls. The menu bar shows '파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)'. The command 'sh if2.sh' has been executed, resulting in the output '거짓입니다'.

```
root@server: ~  
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)  
root@server:~# sh if2.sh  
거짓입니다  
root@server:~#
```


2-4 if문과 case문

- 조건문의 비교 연산자
 - 조건문에서는 문자열 비교와 산술 비교가 가능

표 11-3 문자열 비교 연산자

| 문자열 비교 | 설명 |
|------------------|-------------------------|
| "문자열1" = "문자열2" | 두 문자열이 같으면 참 |
| "문자열1" != "문자열2" | 두 문자열이 같지 않으면 참 |
| -n "문자열" | 문자열이 NULL(빈 문자열)이 아니면 참 |
| -z "문자열" | 문자열이 NULL(빈 문자열)이면 참 |

표 11-4 산술 비교 연산자

| 산술 비교 | 설명 |
|-------------|-----------------------|
| 수식1 -eq 수식2 | 두 수식(또는 변수)이 같으면 참 |
| 수식1 -ne 수식2 | 두 수식(또는 변수)이 같지 않으면 참 |
| 수식1 -gt 수식2 | 수식1이 크면 참 |
| 수식1 -ge 수식2 | 수식1이 크거나 같으면 참 |
| 수식1 -lt 수식2 | 수식1이 작으면 참 |
| 수식1 -le 수식2 | 수식1이 작거나 같으면 참 |
| !수식 | 수식이 거짓이면 참 |

if3.sh

```
1  #!/bin/sh
2  if [ 100 -eq 200 ]
3  then
4      echo "100과 200은 같다."
5  else
6      echo "100과 200은 다르다."
7  fi
8  exit 0
```

```
root@server: ~
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
root@server:~# sh if3.sh
100과 200은 다르다.
root@server:~#
```

2-4 if문과 case문

■ 파일 관련 조건

표 11-5 파일 관련 조건

| 파일 관련 조건 | 설명 |
|----------|--------------------------|
| -d 파일명 | 파일이 디렉터리이면 참 |
| -e 파일명 | 파일이 존재하면 참 |
| -f 파일명 | 파일이 일반 파일이면 참 |
| -g 파일명 | 파일에 set-group-id가 설정되면 참 |
| -r 파일명 | 파일이 읽기 가능이면 참 |
| -s 파일명 | 파일 크기가 0이 아니면 참 |
| -u 파일명 | 파일에 set-user-id가 설정되면 참 |
| -w 파일명 | 파일이 쓰기 가능이면 참 |
| -x 파일명 | 파일이 실행 가능이면 참 |

if4.sh

```
1  #!/bin/sh
2  fname=/lib/systemd/system/cron.service
3  if [ -f $fname ]
4  then
5    head -5 $fname
6  else
7    echo "cron 서버가 설치되지 않았습니다."
8  fi
9  exit 0
```

- 2행: fname 변수에 cron 서버 실행 파일인 /lib/systemd/system/cron.service를 저장
- 3행: fname 변수에 저장된 /lib/systemd/system/cron.service 파일이 일반 파일이면 참이므로 5행을 실행하고, 그렇지 않으면 거짓이므로 7행을 실행
- 5행: fname에 들어 있는 파일의 앞 다섯 행을 출력

```
root@server: ~
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
root@server:~# sh if4.sh
[Unit]
Description=Regular background program processing daemon
Documentation=man:cron(8)

[Service]
root@server:~#
```

2-4 if문과 case문

■ case~esac문

- if문은 참과 거짓, 두 가지 경우에만 사용
- 경우의 수가 셋 이상이라면 if문을 중복해야 하므로 구문이 복잡
- 이럴 때 사용하는 것이 case문

case1.sh

```
1  #!/bin/sh
2  case "$1" in
3    start)
4      echo "시작~~";
5    stop)
6      echo "중지~~";
7    restart)
8      echo "다시 시작~~";
9    *)
10     echo "뭔지 모름~~";
11  esac
12  exit 0
```

```
root@server: ~
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
root@server:~# sh case1.sh stop
중지~~
root@server:~#
```

- 2행: 첫 번째 파라미터 변수(명령 실행 시 추가한 값)인 \$1 값에 따라서 3행, 5행, 7행, 9행으로 분기
- 4행: 3행에서 start)인 경우 실행. 끝에 세미콜론을 2개(;;) 넣어야 함
- 11행: case문의 종료를 나타냄

2-4 if문과 case문

case2.sh

```
1  #!/bin/sh
2  echo "공부가 재미있나요? (yes / no)"
3  read answer
4  case $answer in
5    yes | y | Y | Yes | YES)
6      echo "다행입니다."
7      echo "더욱 열심히 하세요 ^^";;
8    [nN]*)
9      echo "안타깝네요. ππ";;
10   *)
11     echo "yes 아니면 no만 입력했어야죠"
12     exit 1;;
13 esac
14 exit 0
```

- 3행: answer 변수에 입력한 값을 받음
- 5행: 입력된 값이 yes, y, Y, Yes, YES 중 하나이면 6~7행을 실행
- 6행: 실행할 구문이 더 있으므로 끝에 ;;을 넣지 않음
- 7행: 실행할 구문이 없으므로 끝에 ;;을 넣음
- 8행: [nN]*)는 앞에 n 또는 N이 들어가는 모든 단어를 인정한다는 의미
- 12행: 정상적인 종료가 아니므로 exit 1로 종료

root@server: ~

파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)

```
root@server:~# sh case2.sh
공부가 재미있나요? (yes / no)
Y
다행입니다.
더욱 열심히 하세요 ^^
root@server:~# sh case2.sh
공부가 재미있나요? (yes / no)
Nooooooooo
안타깝네요. ππ
root@server:~# sh case2.sh
공부가 재미있나요? (yes / no)
OK
yes 아니면 no만 입력했어야죠
root@server:~#
```

2-4 if문과 case문

■ and, or 관계 연산자

- And의 의미는 -a 또는 &&를, or은 -o 또는 ||를 사용
- -a나 -o는 테스트문([]) 안에서 사용할 수 있는데, 이때 괄호 등의 특수문자 앞에는 w(w)를 넣어야 함

andor.sh

```
1  #!/bin/sh
2  echo "보고 싶은 파일명을 입력하세요."
3  read fname
4  if [ -f $fname ] && [ -s $fname ] ; then
5      head -5 $fname
6  else
7      echo "파일이 없거나, 크기가 0입니다."
8  fi
9  exit 0
```

root@server: ~

파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)

```
root@server:~# sh andor.sh
보고 싶은 파일명을 입력하세요.
/lib/systemd/system/nofile.service
파일이 없거나, 크기가 0입니다.
root@server:~# sh andor.sh
보고 싶은 파일명을 입력하세요.
/lib/systemd/system/cron.service
[Unit]
Description=Regular background program processing daemon
Documentation=man:cron(8)

[Service]
root@server:~#
```

- 4행에서는 입력한 파일 이름이 일반 파일(-f)이고 크기가 0이 아니면(-s) 5행을 실행
- 세미콜론은 앞뒤 구문을 행으로 분리 하는 기능

2-5 반복문

■ for~in문

- 변수에 각각의 값을 넣은 후 do 안에 있는 '반복할 문장'을 실행→값의 개수만큼 반복 실행

```
for 변수 in 값1 값2 값3 ...  
do  
    반복할 문장  
done
```

forin1.sh

```
1  #!/bin/sh  
2  hap=0  
3  for i in 1 2 3 4 5 6 7 8 9 10  
4  do  
5      hap=`expr $hap + $i`  
6  done  
7  echo "1부터 10까지의 합: "$hap  
8  exit 0
```

- 2행: 합계를 누적할 hap 변수를 초기화한
- 3행: i 변수에 1~10을 넣어 5행을 열 번 실행
- 5행: hap에 i 변수의 값을 누적

```
root@server: ~  
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)  
root@server:~# sh forin1.sh  
1부터 10까지의 합: 55  
root@server:~#
```

2-5 반복문

forin2.sh

```
1  #!/bin/sh
2  for fname in $(ls *.sh)
3  do
4    echo "-----$fname-----"
5    head -3 $fname
6  done
7  exit 0
```

```
root@server: ~
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
root@server:~# sh forin2.sh
-----andor.sh-----
#!/bin/sh
echo "보고 싶은 파일명을 입력하세요."
read fname
-----bce.sh-----
#!/bin/sh
echo "무한반복 입력을 시작합니다. (b: break, c: continue, e: exit)"
```

- 현재 디렉터리에 있는 셸 스크립트 파일(*.sh)의 이름과 앞 세 행을 출력
 - 2행: 2행: fname 변수에 ls *.sh 명령의 실행 결과를 하나씩 넣어 4~5행을 실행
 - 4행: 파일 이름을 출력
 - 5행: 파일의 앞 세 행을 출력

2-5 반복문

■ while문

- while문은 조건식이 참인 동안 계속 반복 실행하는 것이 특징

while1.sh

```
1  #!/bin/sh
2  while [ 1 ]
3  do
4      echo "우분투 18.04 LTS"
5  done
6  exit 0
```

```
root@server: ~
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
root@server:~# sh while1.sh
우분투 18.04 LTS
우분투 18.04 LTS
우분투 18.04 LTS
우분투 18.04 LTS
```

- 2행: 조건식 위치에 [1] 또는 [:]이 오면 항상 참이므로 4행을 무한 반복
- 취소하려면 Ctrl + C

2-5 반복문

while2.sh

```
1  #!/bin/sh
2  hap=0
3  i=1
4  while [ $i -le 10 ]
5  do
6    hap=`expr $hap + $i`
7    i=`expr $i + 1`
8  done
9  echo "1부터 10까지의 합 : "$hap
10 exit 0
```

```
root@server: ~
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
root@server:~# sh while2.sh
1부터 10까지의 합 : 55
root@server:~#
```

- 1부터 10까지의 합계를 출력
 - 2행: 합계를 누적할 hap 변수를 초기화
 - 3행: 1부터 10까지 증가하는 i 변수를 선언
 - 4행: i가 10보다 작거나 같으면 6~7행을 실행
 - 6행: hap에 i 변수의 값을 누적
 - 7행: i 변수의 값을 1씩 증가

2-5 반복문

while3.sh

```
1  #!/bin/sh
2  echo "비밀번호를 입력하세요."
3  read mypass
4  while [ $mypass != "1234" ]
5  do
6      echo "틀렸음. 다시 입력하세요."
7      read mypass
8  done
9  echo "통과~~"
10 exit 0
```

```
root@server: ~
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
root@server:~# sh while3.sh
비밀번호를 입력하세요.
3333
틀렸음. 다시 입력하세요.
4444
틀렸음. 다시 입력하세요.
1234
통과~~
root@server:~#
```

- 비밀번호 입력
 - 3행: mypass 변수에 값을 입력받음
 - 4행: mypass 변수의 값이 '1234'가 아니면 6~7행을 실행하고, '1234'이면 while문을 종료
 - 7행: 다시 mypass 변수에 값을 입력받음

2-5 반복문

- until문
 - while문과 용도가 거의 같지만 조건식이 참일 때까지(거짓인 동안) 계속 반복 실행
 - while2.sh를 until문으로 구현하려면 4행을 **until [\$i -gt 10]** 같이 수정
- break문, continue문, exit문, return문
 - break는 반복문을 종료할 때 주로 사용하며, continue는 반복문의 조건식으로 돌아가게 함
 - exit는 해당 프로그램을 완전히 종료
 - 함수 안에서 사용할 수 있는 return은 함수를 호출한 곳으로 돌아가게 함

2-5 반복문

bce.sh

```
1  #!/bin/sh
2  echo "무한반복 입력을 시작합니다. (b: break, c: continue, e: exit)"
3  while [ 1 ] ; do
4      read input
5      case $input in
6          b | B)
7              break;;
8          c | C)
9              echo "continue를 누르면 while의 조건으로 돌아감"
10             continue ;;
11         e | E)
12             echo "exit를 누르면 프로그램(함수)을 완전히 종료함"
13             exit 1;;
14     esac;
15 done
16 echo "break를 누르면 while을 빠져나와 지금 이 문장이 출력됨."
17 exit 0
```

2-5 반복문

```
root@server: ~
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
root@server:~# sh bce.sh
무한반복 입력을 시작합니다. (b: break, c: continue, e: exit)
c
continue를 누르면 while의 조건으로 돌아감
b
break를 누르면 while을 빠져나와 지금 이 문장이 출력됨.
root@server:~#
```

- 3행: 무한 반복을 뜻하며 while [:] 또는 while [true]와 동일
- 5행: 4행에서 입력한 값에 따라 분기
- 6~7행: b 또는 B가 입력되면 7행의 break를 실행(while문을 종료하고 16행을 실행)
- 8~10행: c 또는 C가 입력되면 9~10행의 continue를 실행(3행의 while문 조건식인 [1]로 돌아감)
- 11~13행: e 또는 E가 입력되면 12~13행의 exit를 실행(프로그램 자체)

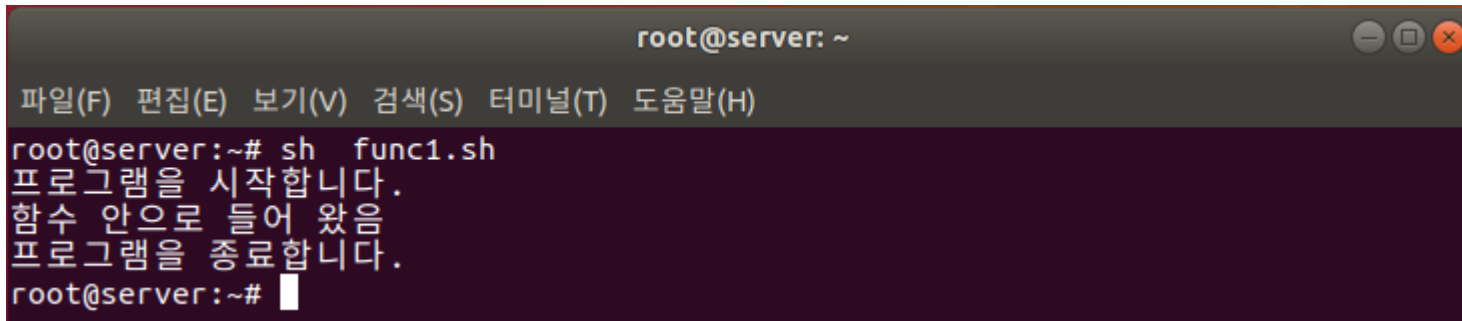
3-1 셸 스크립트 응용 기능

- 사용자 정의 함수(사용자가 직접 함수를 작성하고 호출)

```
함수명 ( ) { -- 함수 정의  
    내용  
}  
함수명      -- 함수 호출
```

func1.sh

```
1  #!/bin/sh  
2  myFunction () {  
3      echo "함수 안으로 들어 왔음"  
4      return  
5  }  
6  echo "프로그램을 시작합니다."  
7  myFunction  
8  echo "프로그램을 종료합니다."  
9  exit 0
```



```
root@server: ~  
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)  
root@server:~# sh func1.sh  
프로그램을 시작합니다.  
함수 안으로 들어 왔음  
프로그램을 종료합니다.  
root@server:~#
```

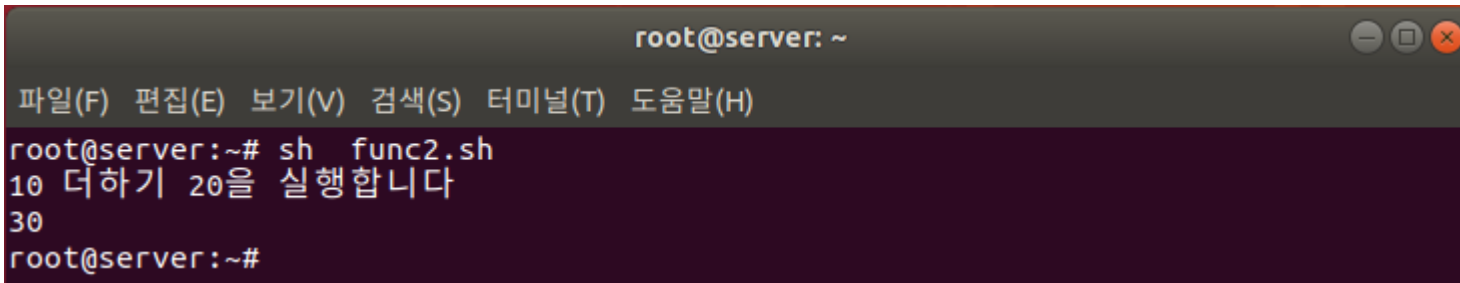
- 2~5행: 함수 정의
- 6행: 여기서부터 프로그램이 시작
- 7행: 함수명을 사용하면 함수를 호출

3-1 셸 스크립트 응용 기능

■ 함수의 파라미터 사용

```
함수명 ( ) {                -- 함수 정의
    $1, $2, ... 등을 사용
}
함수명 파라미터1 파라미터2 ... -- 함수 호출
```

```
func2.sh
1  #!/bin/sh
2  hap () {
3      echo `expr $1 + $2`
4  }
5  echo "10 더하기 20을 실행합니다"
6  hap 10 20
7  exit 0
```



A terminal window titled 'root@server: ~' with a menu bar containing '파일(F)', '편집(E)', '보기(V)', '검색(S)', '터미널(T)', and '도움말(H)'. The terminal shows the command 'sh func2.sh' being executed, followed by the output '10 더하기 20을 실행합니다' and '30'. The prompt returns to 'root@server:~#'.

```
root@server: ~
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
root@server:~# sh func2.sh
10 더하기 20을 실행합니다
30
root@server:~#
```

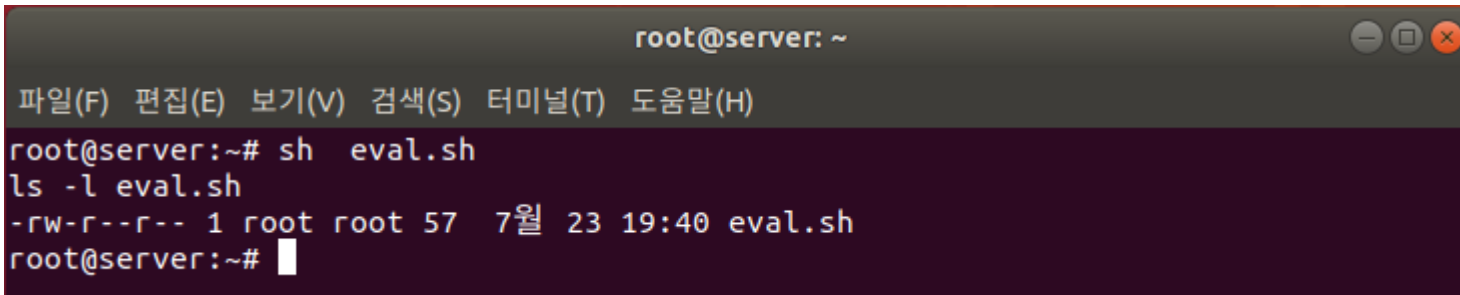
- 함수의 파라미터(인자)를 사용하려면 함수를 호출할 때 뒤에 파라미터를 붙임
- 함수 안에서는 \$1, \$2, ...를 사용
- 3행: 넘겨받은 파라미터 \$1과 \$2를 더한 값을 출력
- 6행: 호출할 때 함수명에 넘겨줄 파라미터를 공백으로 분리하여 차례로 넣음

3-1 셸 스크립트 응용 기능

■ eval

eval.sh

```
1  #!/bin/sh
2  str="ls -l eval.sh"
3  echo $str
4  eval $str
5  exit 0
```



A terminal window titled 'root@server: ~' with standard window controls. The menu bar shows '파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)'. The terminal content shows the execution of 'eval.sh' via 'sh eval.sh', which outputs 'ls -l eval.sh' and then the file details: '-rw-r--r-- 1 root root 57 7월 23 19:40 eval.sh'. The prompt returns to 'root@server:~# '.

```
root@server: ~
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
root@server:~# sh eval.sh
ls -l eval.sh
-rw-r--r-- 1 root root 57 7월 23 19:40 eval.sh
root@server:~#
```

- 문자열을 명령문으로 인식하여 실행
- 3행: str 변수의 값인 'ls -l eval.sh'라는 글자를 그대로 출력
- 4행: str 변수의 값인 'ls -l eval.sh'를 명령으로 인식하여 실행

3-1 셸 스크립트 응용 기능

■ export

exp1.sh

```
1  #!/bin/sh
2  echo $var1
3  echo $var2
4  exit 0
```

exp2.sh

```
1  #!/bin/sh
2  var1="지역 변수"
3  export var2="외부 변수"
4  sh exp1.sh
5  exit 0
```

```
root@server: ~
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
root@server:~# sh exp2.sh
외부 변수
root@server:~#
```

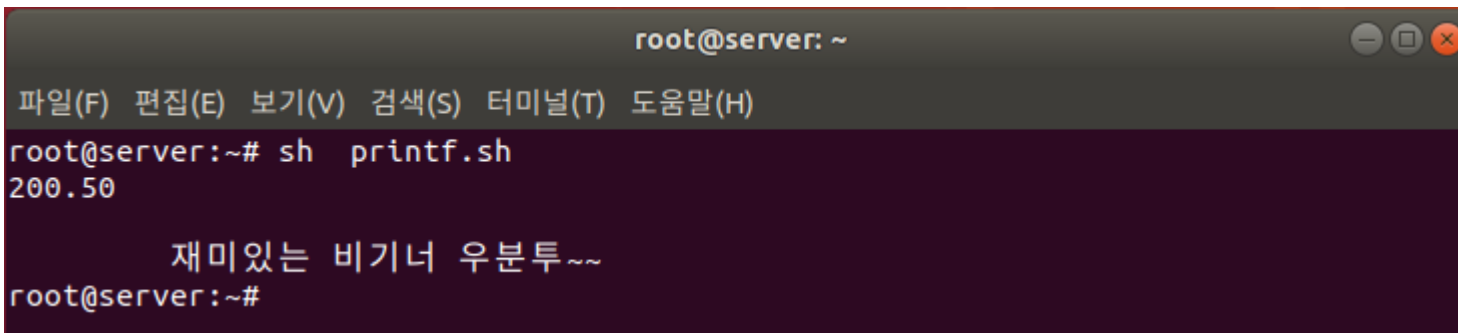
- 특정 변수를 전역 변수로 만들어 모든 셸에서 사용
- exp1.sh 2~3행: var1, var2 변수를 출력
- exp2.sh 2행: var1에 값을 넣고 일반 변수(지역 변수)이므로 현재 프로그램인 exp2.sh에서만 사용
- exp2.sh 3행: var2를 외부 변수로 선언하고 값을 넣음
- exp2.sh 4행: exp1.sh를 실행

3-1 셸 스크립트 응용 기능

■ printf

printf.sh

```
1  #!/bin/sh
2  var1=200.5
3  var2="재미있는 비기너 우분투~~"
4  printf "%5.2f \n\n \t %s \n" $var1 "$var2"
5  exit
```



```
root@server: ~
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
root@server:~# sh printf.sh
200.50

    재미있는 비기너 우분투~~
root@server:~#
```

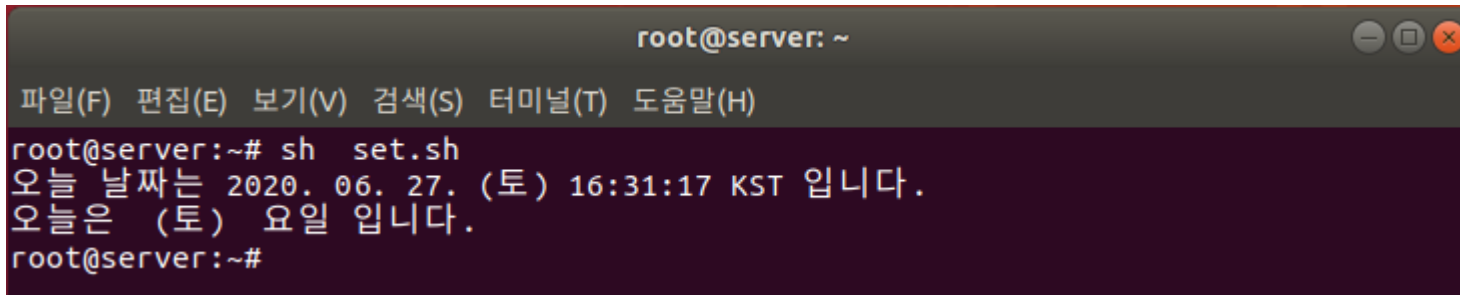
- C 언어의 printf() 함수와 비슷하게 형식을 지정하여 출력
- 3행: 공백이 있으므로 " "로 묶어야 함
- 4행: %5.2f는 총 다섯 자리이며 소수점 아래 두 자리까지 출력하라는 의미
wn은 한 행을 넘기는 개행 문자이고, wt는 Tab 문자이며, %s는 문자열을 출력
\$var2의 경우 값 중간에 공백이 있으므로 변수 이름을 " "로 묶어야 오류 발생 없음

3-1 셸 스크립트 응용 기능

■ set과 \$(명령)

set.sh

```
1  #!/bin/sh
2  echo "오늘 날짜는 $(date) 입니다."
3  set $(date)
4  echo "오늘은 $4 요일 입니다."
5  exit 0
```



A terminal window titled 'root@server: ~' with standard window controls. The menu bar shows '파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)'. The command 'sh set.sh' is entered, followed by the output: '오늘 날짜는 2020. 06. 27. (토) 16:31:17 KST 입니다.' and '오늘은 (토) 요일 입니다.' The prompt returns to 'root@server:~#'.

```
root@server: ~
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
root@server:~# sh set.sh
오늘 날짜는 2020. 06. 27. (토) 16:31:17 KST 입니다.
오늘은 (토) 요일 입니다.
root@server:~#
```

- 리눅스 명령을 결과로 사용하려면 '\$(명령)' 형식 이용
- 결과를 파라미터로 사용하려면 **set** 명령 이용
- 2행: \$(date)는 **date** 명령을 실행한 결과
- 3행: \$(date)의 결과가 파라미터 변수 \$1, \$2, \$3, ...에 저장
- 4행: 네 번째 파라미터인 요일을 출력

3-1 셸 스크립트 응용 기능

■ shift

shift.sh

```
1  #!/bin/sh
2  myfunc () {
3      str=""
4      while [ "$1" != "" ]; do
5          str="$str $1"
6          shift
7      done
8      echo $str
9  }
10 myfunc AAA BBB CCC DDD EEE FFF GGG HHH III JJJ KKK
11 exit 0
```

```
root@server:~# sh shift.sh
AAA BBB CCC DDD EEE FFF GGG HHH III JJJ KKK
root@server:~#
```

- 파라미터 변수를 왼쪽으로 한 단계씩 아래로 시프트(이동)
- 3행: 결과를 누적할 str 변수를 초기화
- 4행: \$1 파라미터가 비어 있지 않은 동안 반복 실행
- 5행: str 변수에 \$1을 추가
- 6행: 전체 파라미터를 왼쪽으로 시프트
- 8행: while문을 빠져나오면 누적한 str 변수를 출력



Thank You
