쉘 스크립트 프로그래밍

Hadoop

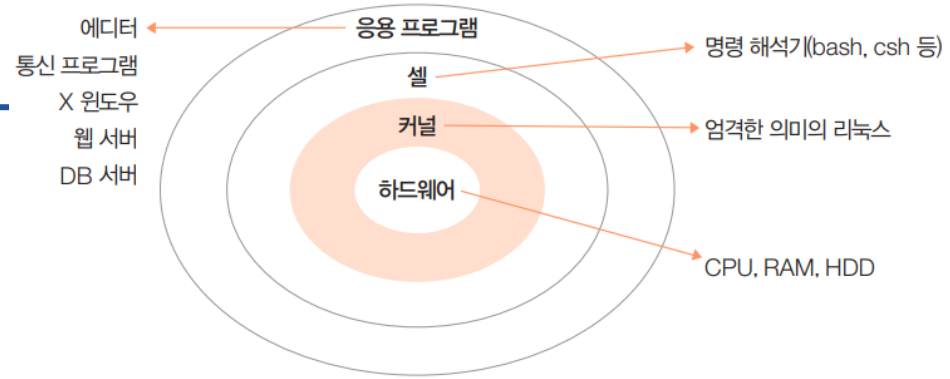
Byeongjoon Noh

powernoh@sch.ac.kr

SCH SOON CHUN HYANG UNIVERSITY

Most of the slides are available on Senseable AI Lab homepage: https://sailab.space/courses/

Shell의 개념



- Linux Shell
  - 명령어와 프로그램을 실행할 때 사용하는 인터페이스
    - 사용자와 커널 사이의 중간 역할
  - 사용자가 입력한 명령을 해석하여 커널에 전달하거나, 커널의 처리 결과를 사용자에게 전달하는 역할
  - Ubuntu에서는 기본적으로 bash (Bourne Again Shell)을 이용함

- Shell의 종류
  - 본쉘, 콘쉘, C쉘, 배시쉘, 로그인쉘, 서브쉘 …

Shell의 개념

- Bash shell의 특징
  - alias (명령 단축) 기능
  - history 기능 (↑ 또는 ↓)
  - 연산 기능
  - Job control 기능
  - 자동 이름 완성 기능 (Tab)
  - 프롬프트 제어 기능
  - 명령 편집 기능

31. 다음 설명에 해당하는 셸의 기능으로 알맞은 것은?

> 기존에 실행한 명령들을 위/아래 방향키를 사용하여 검색 및 편집하여 특정 명령을 반복해서 수행할 수 있다.

① 명령행 완성 기능 ② 명령행 편집 기능
❸ 명령어 히스토리 기능 ④ 명령어 alias 기능

32. 다음 중 현재 사용 가능한 셸 목록 정보가 저장된 파일명으로 알맞은 것은?

① /etc/passwd ❷ /etc/shells
③ /etc/login.defs ④ /etc/default/useradd

33. 다음 설명에 해당하는 셸로 알맞은 것은?

> 1989년 브라이언 폭스가 GNU 프로젝트를 위해 개발한 셸로 명령 히스토리, 명령행 편집 등 다양한 기능을 지원한다.

① ksh ② tcsh
❸ bash ④ dash

쉘의 명령문 처리 방법

- 쉘 명령문의 형식
  - 명령 [옵션] [인자]

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

- 쉘 특수 문자의 종류

메타문자	기능	예제
;	한 줄에 여러 개의 명령 입력	$ date;cal;ls
*	임의의 문자 또는 문자들	$ ls h*
?	임의의 한 문자	$ ls dir?
[ ]	한 문자 위치를 위한 문자의 범위 표시	$ ls [a-f]*
>, >>, <	입출력 방향 전환	$ ls > ls.out
|	명령어 파이핑	$ ls –l /etc | more
~	홈 디렉토리	$ cd ~user1
-	이전 작업 디렉토리	$ cd -
‘ ‘	모든 쉘 문자 무시	$ echo ‘$SHELL’
“ “	$, `, \를 제외한 모든 쉘 문자 무시	$ echo “$SHELL”
` `	쉘 명령 수행	$ echo `date`
\	특수문자 기능 제거	$ echo “\$SHELL”

환경 변수

- 쉘에서는 여러 가지 환경 변수 값을 불러올 수 있음
- 설정된 환경 변수 확인 방법
  - $ echo $[환경변수명]
    - `$ echo $HOSTNAME`
      - 호스트 이름 출력
    - `$ echo $PATH`
      - 라이브러리 경로
- 환경 변수 등록/변경 방법
  - $ export [환경변수명]=값

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 지원 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

쉘 스크립트

- 스크립트 (Script)
  - 응용 프로그램을 제어하기 위한 프로그래밍
  - interpreter라고 불리는 다른 프로그램에 의해 실행되는 프로그램
  - Perl, Java script, Python

- 쉘 스크립트 (Shell script)
  - 쉘이 실행하는 프로그램
  - 유닉스 명령 + 쉘이 제공하는 프로그램 구성 용소
  - 쉘 스크립트 파일 이름은 키워드나, alias, 내장 명령 등과 같은 이름을 쓰지 않는 것이 바람직 함

```
#!/bin/bash
#My First Script Program

echo I love Linux!
pwd
```

쉘 스크립트

- 프로그래밍 언어 vs 스크립트 언어
  - 프로그래밍 언어
    - 소스코드 (*.c, *.java) ➔ compiler (compile) ➔ machine code (*.exe, *.jar)
    - compile: 소스코드를 컴퓨터가 알아들을 수 있는 머신 코드로 바꾸는 작업
    - 장점
      - 소스코드를 머신 코드로 변환해서 실행 ➔ 머신에 최적화되어 빠름
      - 컴파일러에 의한 에러 처리
    - 단점
      - 시스템 종속적
      - 이기종간 이식성 부족 (운영체제에 따라 재 컴파일 필요)

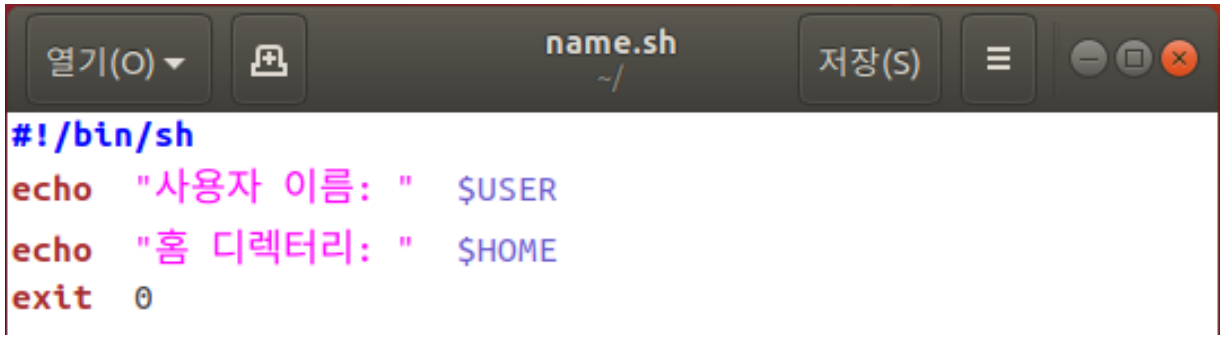
쉘 스크립트

- 프로그래밍 언어 vs 스크립트 언어
  - 스크립트 언어
    - 컴파일러가 존재하지 않고, 인터프리터가 처리
    - 인터프리터: 소스코드를 한 줄씩 읽고 한 줄씩 머신 코드로 바꾸어 실행
    - 장점
      - 시스템 종속적이지 않음
      - 작성한 소스코드를 인터프리터만 있으면 무슨 시스템이든 상관없이 실행 가능
    - 단점
      - 매우 느림
      - 오류처리가 컴파일 언어처럼 엄격하지 않음

쉘 스크립트 작성

- 작성 방법
  - 일반적인 프로그래밍 언어와 비슷하게 변수, 반복문, 제어문 등을 사용
  - 별도로 컴파일 하지 않고 텍스트 파일 형태로 쉘에서 바로 실행
  - 쉘 스크립트는 주로 vi 편집기나 gedit 등에서 바로 작성

- 쉘 스크립트 파일 생성 및 작성
  - `$ vi name.sh` 또는 `$ gedit name.sh`



쉘 스크립트 작성

열기(O) name.sh ~/ 저장(S)

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

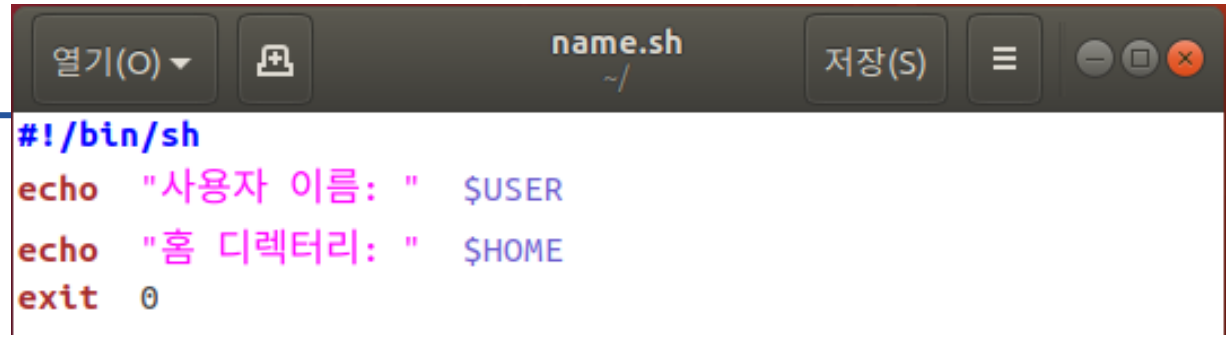
- 작성 방법
  - 주의 사항
    - 1) 첫 행에 반드시 #!/bin/sh로 배시쉘을 사용하겠다고 명시해야 함!
      - 스크립트를 실행할 프로그램을 지정하는 것
      - #!/bin/more, #!/bin/rm 등 다양한 명명

```
#!/bin/more
# test_sharp:스스로를출력하는스크립트
# 이 스크립트를 실행시키면 자기 자신을 화면에 출력합니다.
# 주석문도 모두 출력되지요.

echo 이 문장도 출력됩니다.
```

```
#!/bin/rm
# test_sharp2:자기자신을지우는스크립트
# 이 스크립트를 실행시키면 이 파일이 지워집니다.

echo 이 부분은 절대로 출력되지 않을 겁니다.
```

쉘 스크립트 작성

- 작성 방법
  - 주의 사항
    - 2) exit 0 ➔ 종료 코드 (0은 성공을 의미)
      - 종료 상태는 $? 변수에 저장됨

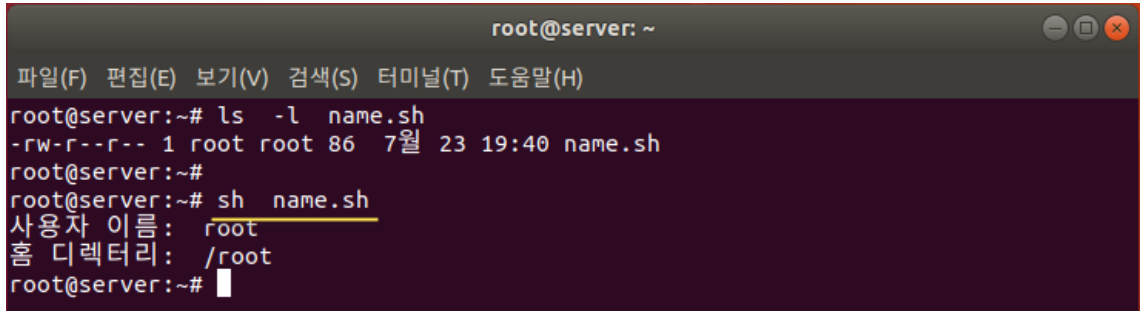



```
#!/bin/bash
#test_exit : exit and $?

echo Test Script
exit 20
```

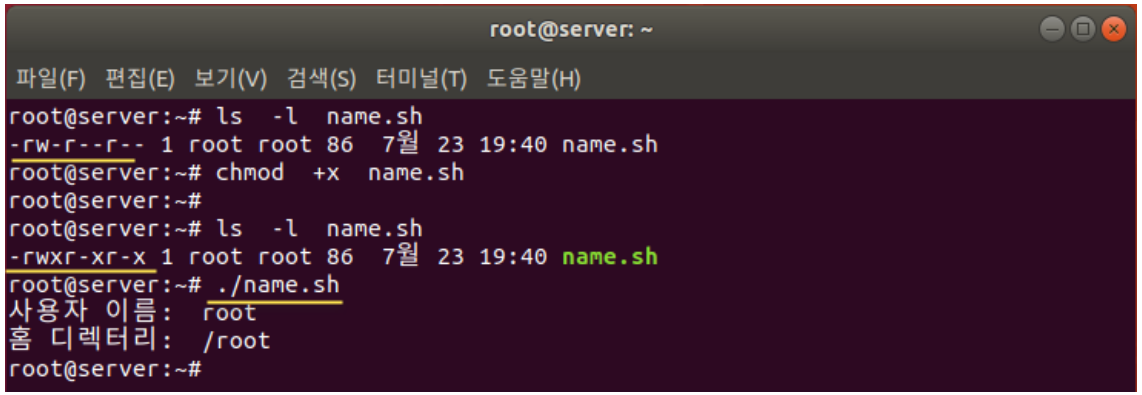
```
$ ./test_exit
Test Script
$ echo $?
20
```

쉘 스크립트 실행

- 실행 방법
  - sh 명령으로 실행
    - $ sh 스크립트파일 명령으로 실행하는 방법
    - 쉘 스크립트 파일의 속성 변경 필요 없이 바로 실행 가능



  - '실행 가능' 속성으로 변경 후 실행
    - $ chmod +x 파일명 또는 $ chmod 755 파일명
    - 실행은 ./파일명 형태로 실행



쉘 스크립트 파일 내용

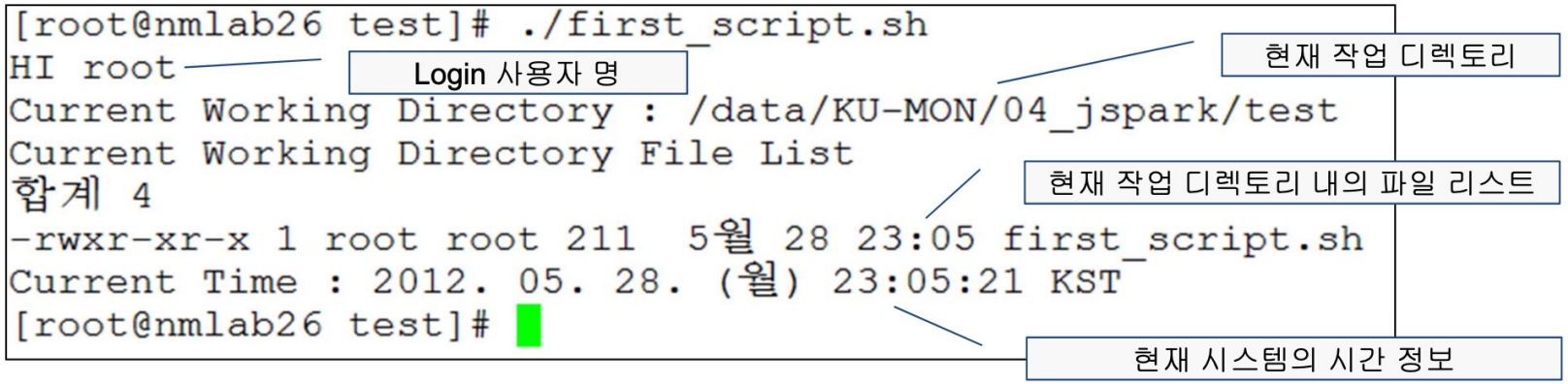
- 쉘 명령어
  - 쉘이 실행할 수 있는 모든 명령어 사용 가능
  - 여러 명령을 반복 수행해야 할 때 파일로 작성하여 실행
  - 예시) find_script

```
#!/bin/bash
# find_script : /bin, /usr/bin에 있는 쉘 스크립트 검색

cd /bin
file * | grep script

cd /usr/bin
file * | grep script
```

```
$./find_script
alsaunmute:         POSIX shell script text executable
gunzip:             POSIX shell script text executable
redhat_lsb_init:    Bourne-Again shell script text executable
```

쉘 스크립트 작성 실습

- 아래와 같이 출력되는 스크립트를 작성해보세요.



쉘 변수

- 변수값 활용

형식	의미
$name	**name**의 값으로 대체
${name}	**name**의 값으로 대치. 변수 이름이 다른 구문과 인접해 있을 때 사용
${name-word}	**name**이 정의되어 있으면 그 값을, 그렇지 않으면 **word**값 사용
${name+word}	**name**이 정의되어 있으면 **word** 값을 사용
${name=word}	**name**이 정의되지 않았으면 **word**를 대입하고 그 값 사용
${name?word}	**name**이 정의되어 있으면 그 값을 사용하고 그렇지 않으면 **word** 출력 후 종료

```
$ test=test
$ echo $test
test
$ echo ${test}
test
$ echo ${test-word}
test
$ echo ${test1-word}
word
$ echo ${test+word}
word
$ echo ${test=word}
test
$ echo ${test1=word}
word
$ echo $test1
word
$ echo ${test?word}
test
$ echo ${test2?word}
-bash: test2: word $
```

쉘 변수 문자열 처리

- 변수의 값이 문자열 일 때 문자열 내 패턴을 찾아 일부분을 제거하는 방법 출력

표현식	기능
${vairable%pattern}	**variable**의 뒤부터 패턴과 일치하는 첫번째 부분을 찾아서 제거
${variable%%pattern}	**variable** 값의 뒤부터 패턴과 일치하는 가장 큰 부분을 찾아서 제거
${variable#pattern}	**variable** 값의 앞부터 패턴과 일치하는 첫번째 부분을 찾아서 제거
${variable##pattern}	**variable** 값의 앞부터 패턴과 일치하는 가장 큰 부분을 찾아서 제거

```
$ path1="/usr/bin/local/bin"
$ echo ${path1%/bin}
/usr/bin/local
$ echo ${path1%%/bin*}
/usr
```

```
$ path2="/export/home/user1/.profile"
$ echo ${path2#/export/home}
/usr1/.profile
$ echo ${path2##*/}
.profile
```

(참고) 변수 속성 지정

- typeset
  - 변수의 속성
    - 대소문자, 폭, 정렬 방향, 읽기 전용, 정수타입 등

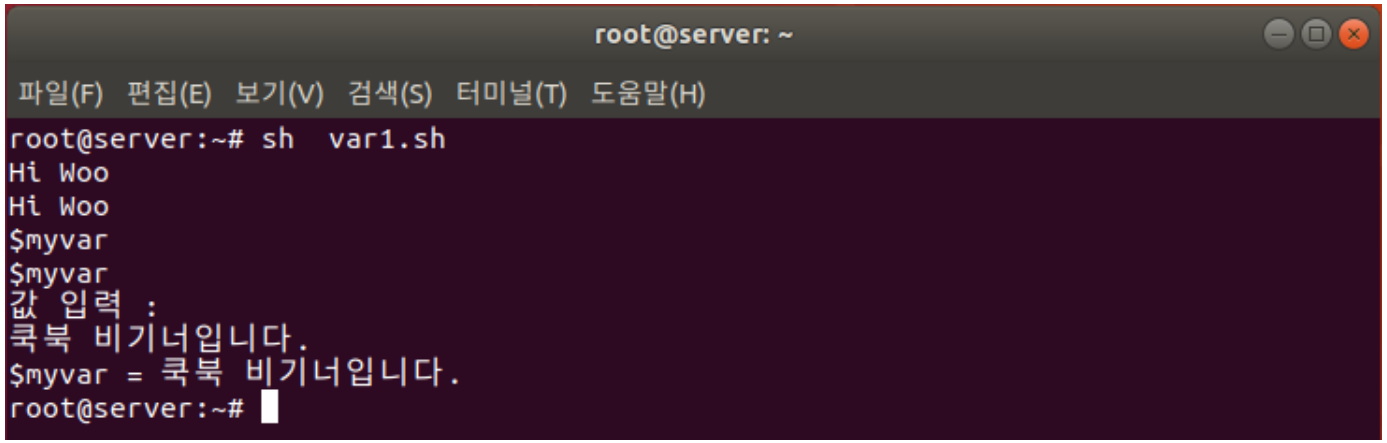
명령	기능
typeset	모든 변수 출력
typeset –x	**export** 된 모든 변수 출력
typeset a b c	지역변수 **a b c** 정의
typeset –r d	변수 **d**를 읽기 전용으로 지정
typeset –i num	**num** 변수에는 정수만 저장
typeset –u name	**name**의 값을 대문자로 변환하여 저장
typeset –l name	**name**의 값을 소문자로 변환하여 저장
typeset –L# name	**name**의 값의 처음 #개 문자만 남기고 삭제
typeset –R# name	**name** 값의 마지막 #개 문자만 남기고 삭제
typeset –z# name	**name**을 #개 문자로 구성된 문자열을 만들고 끝에 **null**문자 추가

변수의 입력과 출력

- $가 포함된 글자를 출력할 때는 ‘ ' 로 묶거나, 앞에 ₩를 붙임

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
```



숫자 계산

- 변수 값을 +, -, *, / 등으로 연산할 때는 expr 키워드 사용
  - 단, 수식과 함께 백틱(`)으로 묶어야 함
    - 백틱 (또는 백쿼트) ` ➔ 숫자 1왼쪽에 있음 (Mac 사용자: 영어모드에서 물결)
  - 수식에 괄호 또는 곱셈(*)를 사용하려면 반드시 앞에 ₩ 붙임

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
```

3행: 문자열로 취급하며 모두 붙여서 써야 함

5행: 숫자로 취급하여 계산하며 각 단어를 띄어쓰기해야 함

```
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:~#
```

파라미터 변수

- 파라미터 변수 = $0, $1, $2, ... 형태
  - 실행하는 명령의 각 부분을 변수로 지정
  - 예시
    - `$ apt-get -y install gftp`
  - $* ➔ 명령 전체

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:~#
```

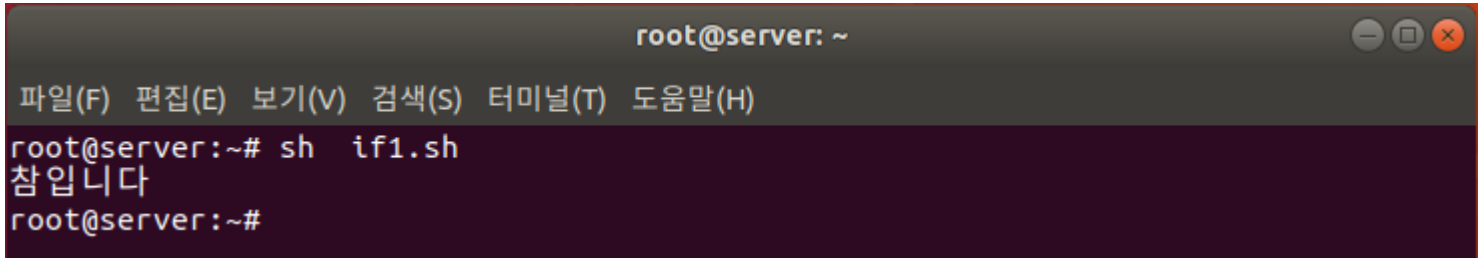
if문과 case문

- 기본 if문
  - 주의: [조건] 안의 각 단어 사이에 공백이 있어야 함

```
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
```



  - = 같음 비교 (대입 아님)
  - != 다름 비교 (같지 않음)

if문과 case문

- if~else문

```
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
```

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

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:~#
```

if문과 case문

- 파일 관련 조건

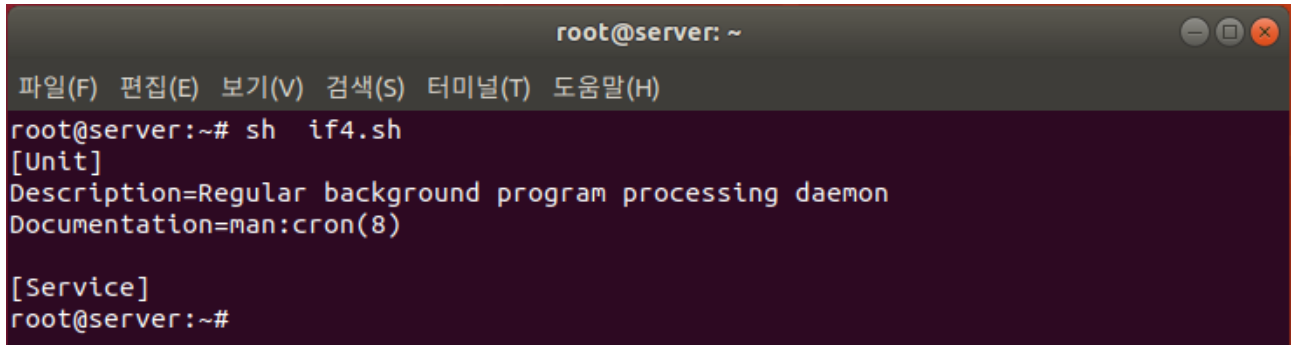
표 11-5 파일 관련 조건

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

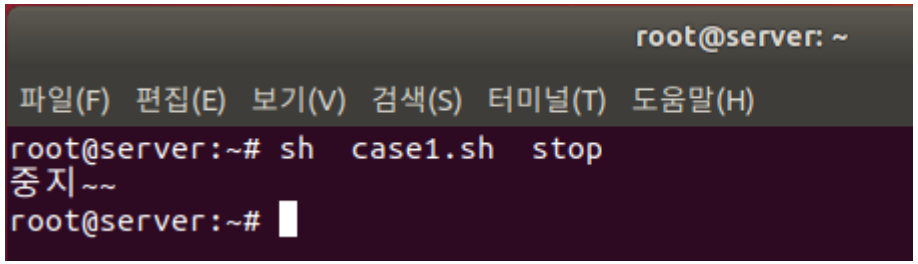


if문과 case문

- case-esac문

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
```



- 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

if문과 case문

- case-esac문

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. ㅠㅠ";;
10   *)
11     echo "yes 아니면 no만 입력했어야죠"
12     exit 1;;
13 esac
14 exit 0
```

```
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)
Noooooooo
안타깝네요. ㅠㅠ
root@server:~# sh  case2.sh
공부가 재미있나요? (yes / no)
OK
yes 아니면 no만 입력했어야죠
root@server:~#
```

- 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

if문과 case문

- and, or 관계 연산자
  - and: -a 또는 &&
  - or: -o 또는 ||

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

반복문

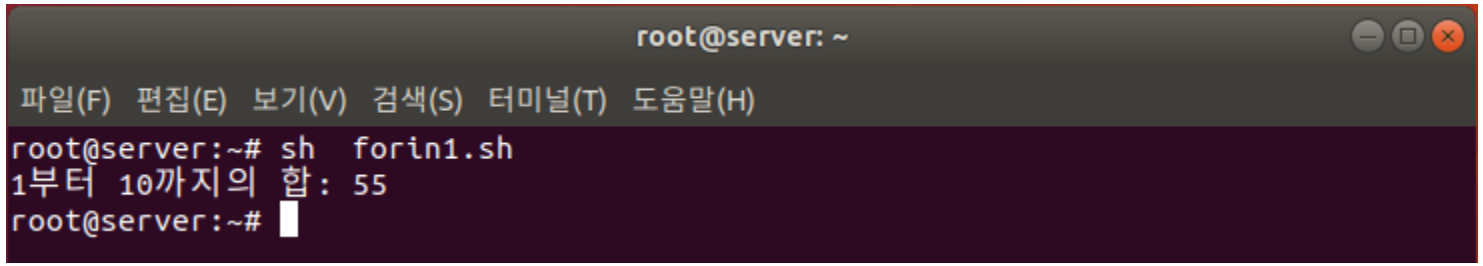
- for~in문
  - 변수에 각각의 값을 넣은 후 do 안에 있는 '반복할 문장'을 실행 ➔ 값의 수 만큼 반복 실행

```
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



반복문

- for~in문

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)"
```

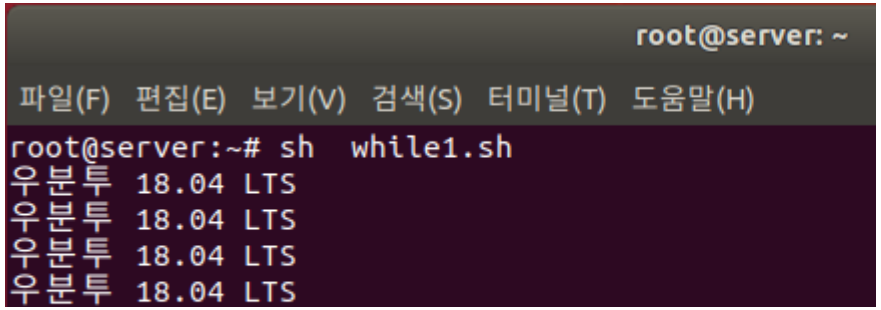
- 현재 디렉터리에 있는 셸 스크립트 파일(*.sh)의 이름과 앞 세 행을 출력
- 2행: 2행: fname 변수에 ls *.sh 명령의 실행 결과를 하나씩 넣어 4~5행을 실행
- 4행: 파일 이름을 출력
- 5행: 파일의 앞 세 행을 출력

반복문

- while문

while1.sh

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



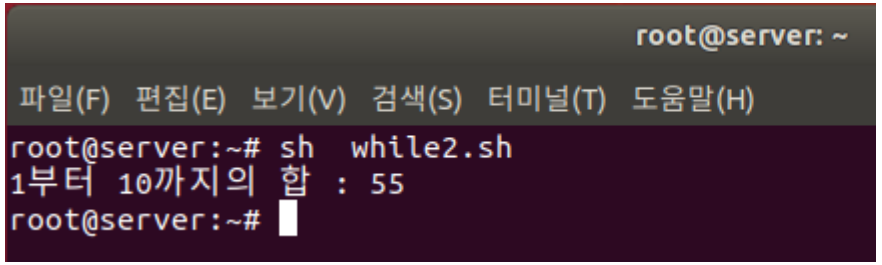
- 2행: 조건식 위치에 [1] 또는 [:]이 오면 항상 참이므로 4행을 무한 반복
- 취소하려면 Ctrl + C

반복문

- while문

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
```



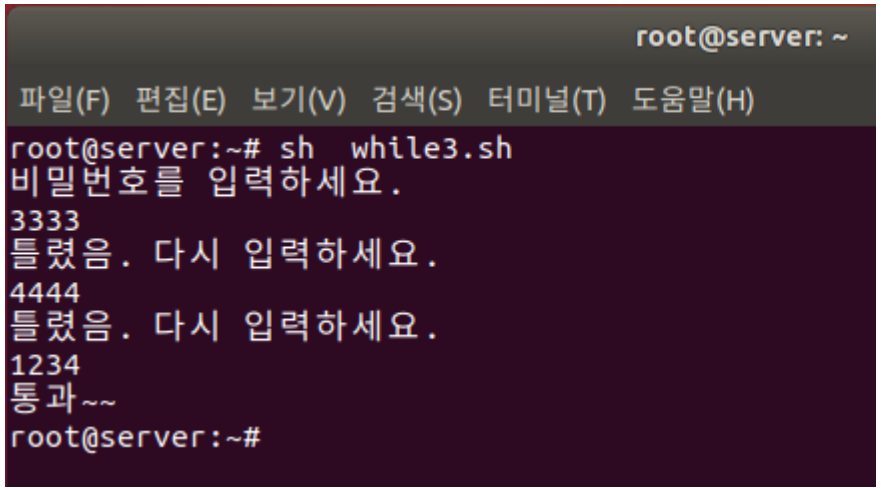
- 1부터 10까지의 합계를 출력
- 2행: 합계를 누적할 hap 변수를 초기화
- 3행: 1부터 10까지 증가하는 i 변수를 선언
- 4행: i가 10보다 작거나 같으면 6~7행을 실행
- 6행: hap에 i 변수의 값을 누적
- 7행: i 변수의 값을 1씩 증가

반복문

- while문

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
```



- 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

반복문

- until문
  - while문과 용도가 같지만, 조건식이 참일 때까지(거짓인 동안) 계속 반복실행
  - while2.sh를 until문으로 구현하면 4행을 until [$i –gt 10] 와 같이 수정

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
```

- break문, continue문, exit문, return문
  - ~~너들이 아는 그거야~~
  - 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

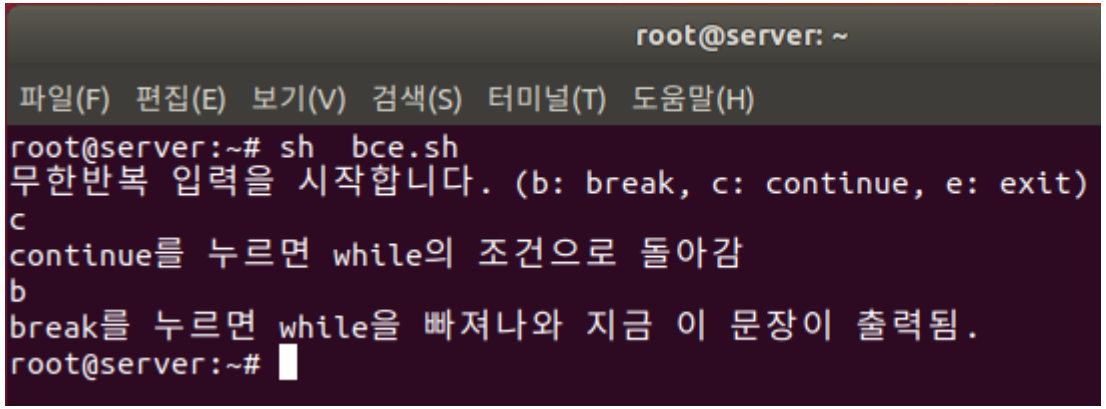
반복문

- break문, continue문, exit문, return문

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
```

- 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)



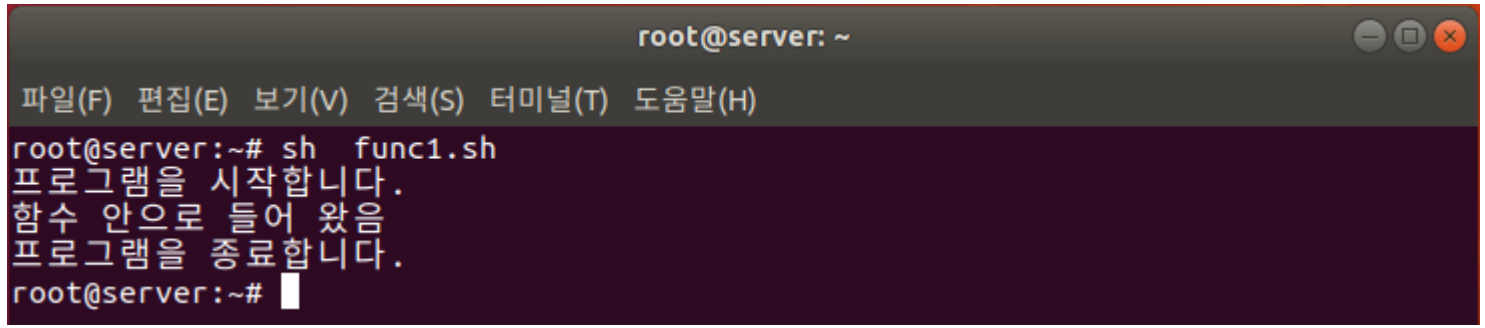
쉘 스크립트 응용

- 사용자 정의 함수

```
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
```



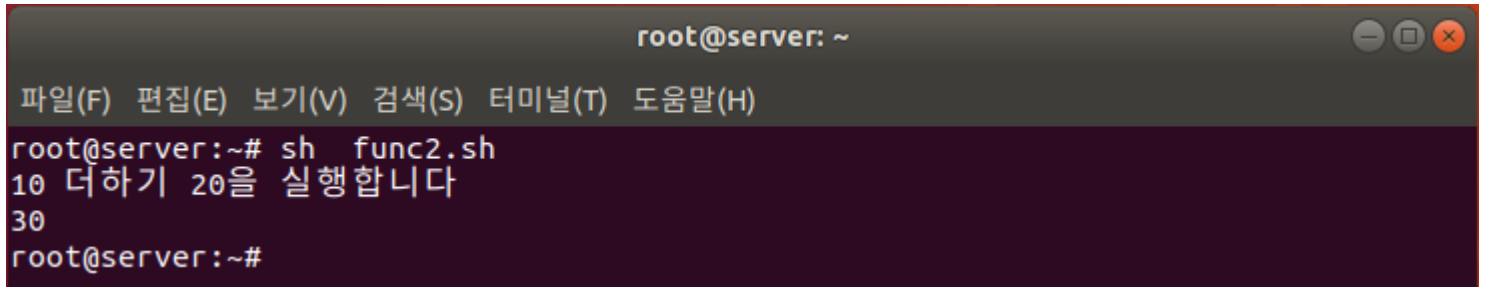
쉘 스크립트 응용

- 사용자 정의 함수 - 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
```



- 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

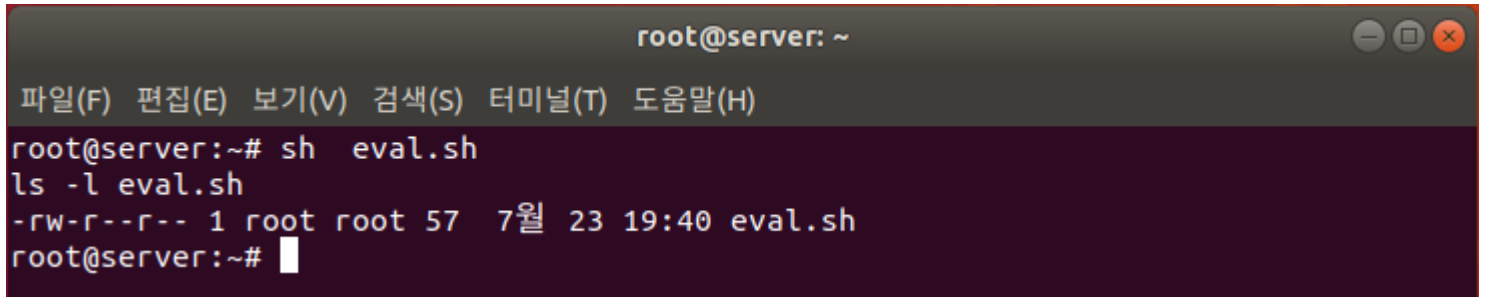
쉘 스크립트 응용

- eval
  - 문자열을 명령문으로 인식하여 실행

eval.sh

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

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



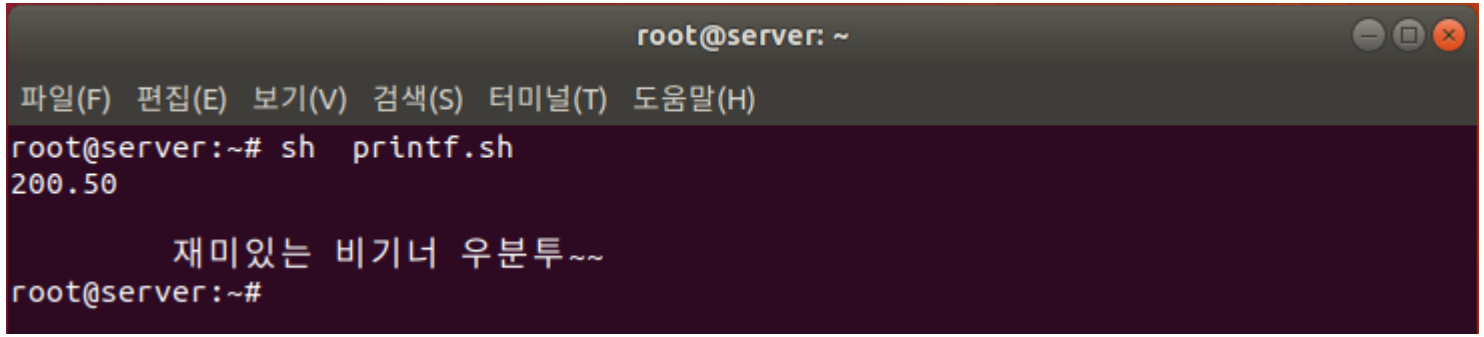
쉘 스크립트 응용

- printf
  - C언어의 printf()와 유사하게 동작

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
```

- 3행: 공백이 있으므로 “ ”로 묶어야 함
- 4행: %5.2f는 총 다섯 자리이며 소수점 아래 두 자리까지 출력하라는 의미
  ₩n은 한 행을 넘기는 개행 문자이고, ₩t는 Tab 문자이며, %s는 문자열을 출력
  $var2의 경우 값 중간에 공백이 있으므로 변수 이름을 “ ”로 묶어야 오류 발생 없음



End of slide