# Linux 1

- 기본 Shell Programing

# INDEX

# Shell 기본

# Shell 기본

## ▶Shell 이란

셸은 사용자가 입력한 명령을 해석해 커널에게 전달하거나, 커널의 처리 결과를 사용자에게 전달하는 역할을 한다.

## ▶ bash Shell 특징

Alias 기능(명령어 단축 기능)
History 기능(↑, ↓)
연산 기능
Job control 기능
자동 이름 완성 기능(Tab)
프롬프트 제어 기능
명령 편집기능

# Shell 기본

## ▶환경변수

쉘은 여러가지 환경 변수 값을 갖는다. 설정된 환경 변수는 'echo $환경변수이름' 형식으로 명령어를 실행하면 확인할 수 있다

| 환경 변수 | 설명 | 환경 변수 | 설명 |
|---|---|---|---|
| HOME | 핸재 사용자의 홈디렉토리 | PATH | 실행 파일을 찾는 디렉토리 경로 |
| LANG | 기본 지원되는 언어 | PWD | 사용자의 현재 작업 디렉터리 |
| TERM | 로그인 터미널 타입 | SHELL | 로그인해서 사용하는 셀 |
| USER | 현재 사용자의 이름 | DISPLAY | X 디스플레이 이름 |
| COLUMNS | 현재 터미널의 컬럼 수 | LINES | 현재 터미널 라인 수 |
| PS1 | 1차 명령 프롬프트 변수 | PS2 | 2차 명령 프롬프트 |
| BASH | bash 쉘의 경로 | BASH_VERSION | bash 버전 |
| HISTFILE | 히스토리 파일의 경로 | HISTSIZE | 히스토리 파일에  저장되는 개수 |
| HOSTNAME | 호스트의 이름 | USERNAME | 현재 사용자 이름 |
| LOGNAME | 로그인 이름 | LS_COLORS | 'ls'명령어의 확장자 색상 옵션 |
| MAIL | 메일을 보관하는 경로 | OSTYPE | 운영체제 타입 |

# ▶쉘 스크립트 프로그래밍

리눅스의 쉘 스크립트는 C언어와 유사한 방법으로 프로그래밍 할 수 있다. 왜냐하면
리눅스의 대부분을 C언어로 작성했기 때문이다.

쉘 스크립트도 일반적인 프로그래밍 언어와 비슷하게 변수, 반복문, 제어문 등을 사용할 수 있다.
또한 별도로 컴파일하지 않고 텍스트 파일 형태로 쉘에서 바로 실행할 수 있다.
그래서 쉘 스크립트는 주로 vi 에디터나 gedit으로 작성하는 편이다

# Shell 기본

▶기본 쉘스크립트(name.sh)

```
#!/bin/sh                                  // 사용쉘 지정
echo “사용자 이름 :” $USERNAME             // 문자열로 출력($USERNAME 함수값을)
echo “호스트 이름 :” $HOSTNAME             // 문자열로 출력($HOSTNAME 함수값을)
exit 0                                     // 종료 코드를 반환
                                              0 : 성공(기본) , 1 : 실패
```

▶ ‘실행 가능’ 속성으로 변경

```
chmod +x name.sh
./name.sh
```

# 변 수

# 변 수

## ▶변수

변수는 필요한 값을 계속 변경해 저장한다는 개념이다.
쉘 스크립트의 구조는 변경할 필요가 없는데 설정해야 하는 값이 상황에 따라 다르다면
변수에 필요한 값을 계속 바꿔가는방법으로 프로그래밍해 다양한 상황에 대처할 수 있다

## ▶ 변수의 기본

쉘 스크립트에서는 변수를 사용하기 전에 미리 선언하지 않으며,
처음 변수에 값이 할당되면 자동으로 변수가 생성된다
변수에 넣는 모든 값은 문자열로 취급한다. 즉, 숫자를 넣어도 문자로 취급한다
변수 이름은 대소문자를 구분한다. 즉, $aa라는 변수 이름과 $AA라는 변수 이름은 다르다
변수를 대입할때 '='좌우에는 공백이 없어야 한다

# 변 수

## ▶변수의 예

testval = Hello -> 오류! '=' 앞 뒤에 공백이 있다.
testval=Hello
testval=Yes Sir -> 오류! 값의 공백은 ""로 묶어야 한다
testval="YES Sir"
testval=7+5 -> 정상이지만 "7+5"라는 문자열로 인식한다

## ▶변수의 입력과 출력

$ 라는 문자가 들어간 글자를 출력하려면 ''로 묶어주거나 앞에 '\'를 붙여야한다
또한 ""로 변수를 묶어도 되고, 묶지 않아도 된다

# 변 수

## ▶ 변수의 입력과 출력 - 1

입력 코드 (var1.sh)

```
 1 #!/bash/sh
 2 myvar="Hi Woo"
 3 echo $myvar
 4 echo "$myvar"
 5 echo '$myvar'
 6 echo \$myvar
 7 echo 값 입력:
 8 read myvar
 9 echo '$myvar' = $myvar
10 exit 0
```

출력값 (sh var1.sh)

```
Hi Woo
Hi Woo
$myvar
$myvar
값 입력:
hello world
$myvar = hello world
```

참고 : '$변수' 와 "$변수" 는 동일하게 인식한다. 하지만 변수에 입력된 값에 공백이 포함될수 있다면 "$변수" 형식을 사용하는 것이 공백으로 인해 발생할 수 있는 논리 오류를 방지하는데 도움이 된다

# 변 수

## ▶ 숫자 계산

변수에 넣은 값은 모두 문자열로 취급한다
만약 변수에 들어 있는 값에 '+','-','*','/' 등의 연산을 하려면 'expr'키워드를 사용한다
단, 수식과 함께 꼭 키보드 숫자 1 왼쪽에 있는 역따옴표(`)로 묶어야한다
수식에 괄호를 사용하려면 그 앞에 꼭 역슬래시(\)를 붙여줘야한다.
또 '+','-','/'와 달리 곱하기(*) 기호도 예외적으로 앞에 역슬래시(\)를 붙여줘야 한다

## ▶ 실습 코드(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
```

## ▶ 실행

```
sh ./numcalc.sh

100+200   //문자열 취급
300       //각 단어마다 띄어쓰기할 것
60        //괄호와 '*'앞에는 \를 붙인다
```

# 변 수

## ▶ 파라미터 변수

파라미터 변수는 '$0','$1','$2' 등의 형태를 갖는다.
이는 실행하는 명령의 부분 하나하나를 변수로 지정한다는 의미이다
예를 들어 'yum -y install gftp를 실행한다고 가정하면 파라미터 변수는 다음과같이
지정할수 있다

[파라미터 변수]

| 명령어 | yum | -y | install | gftp |
|---|---|---|---|---|
| 파라미터 변수 | $0 | $1 | $2 | $3 |

# 변 수

## ▶ 파라미터 변수(paravar.sh)

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

## ▶ 파라미터 변수(sh paravar.sh aa bb )

```
sh paravar.sh aa bb
실행파일 이름은 <paravar.sh>이다.
첫번째 파라미터는 <aa>이고, 두번째 파라미터는 <bb>다
전체 파라미터는 <aa bb>다
```

# if문과 case 문

# if~fi문

## ▶ 문법

```
if [조건]
then
 참일 경우 실행
fi
```

-주의-
[조건] 사이의 각 단어에는 모두 공백이 있어야 한다

# if~fi문

▶ 예제(fi1.sh)

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

▶ 확인

```
 sh if1.sh
참입니다
```

# if~fi문

## ▶ 소스설명

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

[] 사이에는 참과 거짓을 구분하는 조건식이 들어간다. '='는 문자열이 같은지를 비교하며, '!='는 문자열이 같지 않은지 비교한다. if1.sh에서는 조건식이 참이므로 4행을 실행한다
또 [] 대신 test라는 키워드를 사용할수도 있다
예 )  if test "woo" = "woo"

# if~fi문

## ▶ if~else 문

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

## ▶ 예제 코드

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

## ▶ 확인

```
[root@localhost test]# sh if2.sh
거짓입니다.
```

# if~fi문

## ▶ 조건문에 들어가는 비교 연산자

조건문에 들어가는 비교연산자에는 문자열 비교 연산자와 산술 비교 연산자가 있다

[문자열 비교 연산자]

| 문자열 비교 | 결과 |
|---|---|
| “문자열1” = “문자열2” | 두 문자열이 같으면 참 |
| “문자열1” != “문자열2” | 두 문자열이 같지 않으면 참 |
| -n “문자열” | 문자열이 Null(빈문자열)이 아니면 참 |
| -z “문자열” | 문자열이 Null(빈 문자열)이면 참 |

# if~fi문

## ▶ 조건문에 들어가는 비교 연산자

조건문에 들어가는 비교연산자에는 문자열 비교 연산자와 산술 비교 연산자가 있다

[산술 비교 연산자]

| 문자열 비교 | 결과 |
|---|---|
| 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 크다면 참 |
| 수식1 -ge 수식2 | 수식1이 크거나 같으면 참 |
| 수식1 -lt 수식2 | 수식1이 작으면 참 |
| 수식1 -le 수식2 | 수식1이 작거나 같으면 참 |
| !수식 | 수식이 거짓이라면 참 |

# if~fi문

▶ 예제 코드

```
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
```

▶ 실행 결과

```
sh if3.sh

100과 200은 다르다
```

# if~fi문

## ▶ 파일과 관련된 조건

| 파일조건 | 결과 |
|---|---|
| -d 파일이름 | 파일이 디렉터리면 참 |
| -e 파일이름 | 파일이 존재하면 참 |
| -f 파일이름 | 파일이 일반 파일이면 참 |
| -g 파일이름 | 파일에 set-group-id가 설정되면 참 |
| -r 파일이름 | 파일이 읽기 가능이면 참 |
| -s 파일이름 | 파일 크기가 0이 아니면 참 |
| -u 파일이름 | 파일에 set-user-id가 설정되면 참 |
| -w 파일이름 | 파일이 쓰기 가능 상태이면 참 |
| -x 파일이름 | 파일이 실행 가능 상태이면 참 |

# if~fi문

## ▶ 예제 코드

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

## ▶ 실행 결과

[웹서버 미설치시]
sh if4.sh
웹서버가 설치되지 않았습니다.

[웹서버 설치시]
yum install -y httpd

sh if4.sh [head -5 $fname]

```
[Unit]
Description=The Apache HTTP Server
After=network.target remote-fs.target nss-
lookup.target
Documentation=man:httpd(8)
Documentation=man:apachectl(8)
```

# case~esac 문

## ▶ case~esac 문

if 문은 참과 거짓이라는 두가지 경우만 사용할 수 있다(이를 '2증 분기'라 한다)
그런데 여러가지 경우의 수가 있다면 if문을 계속 중복해서 사용해야 하므로 구문이 복잡해진다
이때 사용하는 것이 case문이다('다중 분기'라한다)

## ▶ case~esac 문(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
```

## ▶ 확인

```
sh case1.sh stop
중지~~
```

-소스 설명-

| | |
|---|---|
| 2행 | 첫번째 파라미터 변수(명령 실행 시 추가한 값)인 '$1'값에 따라서<br>3행,5행,7행,9행으로 분기함<br>'start)','stop)','resetart)' 이외의 값은 모두 9행'*'부분으로 분기한다 |
| 4행 | 3행에서 'start)'일 경우에 실행된다.<br>주의할 점은 맨 뒤에 세미콜론을 2개(';;') 붙여서 써야 한다 |
| 9행 | 그 외의 것들이 모두 해당한다 |
| 11행 | case문 종료를 표시한다 |

# case~esac 문

## ▶ case 문의 활용(case2.sh)

[예제 코드]

```
1    #!/bin/sh
2    echo "리눅스가 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 ㅜㅜ";;
10
11   *)
12     echo "Yes 아니면 No만 입력해야합니다"
13     exit 1;;
14   esac
15   exit 0
```

[확인]

```
[root@localhost test]# sh case2.sh
리눅스가 재미있나요?(yes/no)
yes
다행입니다
더욱 열심히 하세요 !!
[root@localhost test]# sh case2.sh
리눅스가 재미있나요?(yes/no)
no
안타깝네요 ㅜㅜ
[root@localhost test]# sh case2.sh
리눅스가 재미있나요?(yes/no)
wefw
Yes 아니면 No만 입력해야합니다
```

# case~esac 문

## ▶ 소스 설명

[예제 코드]

```
1     #!/bin/sh
2     echo "리눅스가 재미잇나요?(yes/no)"
3     read answer                                  //answer 변수에 입력한 값을 받는다
4     case $answer in
5     yes | y | Y | Yes | YES)                     //yes,y,Y,Yes,YES 중 하나면 6~7행을 실행한다
6       echo "다행입니다"                         //실행할 구문이 더 있으므로 ';;'를 붙이지 않는다
7       echo "더욱 열심히 하세요 !!";;            //실행할 구문이 더 없으므로 ';;'를 붙인다
8     [nN]*)                                       //앞에 n또는 N이 들어가는 모든 단어를 인정한다
9       echo "안타깝네요 ㅜㅜ";;
10
11    *)
12      echo "Yes 아니면 No만 입력해야합니다"
13      exit 1;;                                   //정상적인 종료가 아니므로 exit 1로 종료한다
14    esac                                           (꼭 해야 하는 사항은 아님)
15    exit 0
```

# AND,OR관계 연산자

# AND,OR관계 연산자

## ▶ AND,OR 관계 연산자

조건문에서는 and와 or의 의미를 갖는 관계 연산자를 사용할 수 있다.
and는 '-a' 또는 '&&'를 사용하며, or은 '-o' 또는 '||'을 사용한다
'-a'나 '-o'는 테스트문([])안에서 사용할 수 있는데,
이때 괄호 등의 특수 문자 앞에는 역슬러시(\)를 붙여줘야한다

## ▶ 예제 코드(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 0 입니다."
8 fi
9 exit 0
```

## ▶ 예제 코드(andor.sh)

```
sh ./andor.sh
보고 싶은 파일명을 입력하세요.
/etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
```

# AND,OR관계 연산자

## ▶ 예제코드(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 0 입니다."
8 fi
9 exit 0
```

## ▶ 소스 설명

```
// 입력한 파일이름이 일반 파일('f')이고, 크기가 0이 아니라면
// 5행을 실행한다
// then 구문은 다음 줄에 작성해도 되며 세미콜론(';')이후에
// 작성해도 된다
// 세미콜론은 앞뒤 구문을 행으로 분리해주는 기능이다
// 이 구문은 'if[ \( -f $fname \) -a \( -s $fname \) ]; then
// 과 동일하다
```

# For ~ in 문

# for~in 문

## ▶for ~ in 문

for~in 문은 다음 형식을 보면 변수에 각각의 값을 넣은 후 do 안에 있는 '반복할 문장을' 실행한다
그러므로 값의 개수만큼 반복 실행한다

## ▶기본 문법

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

# for~in 문

▶ 예제 코드(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
```

▶ 확인

```
sh forin.sh

1부터 10까지의 합:55
```

# for~in 문

▶ 예제 코드(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
```

▶ 소스 설명

```
// 합계를 누적할 변수를 0으로 초기화한다
// i변수에 1~10까지 반해 넣으면서 5행을 10회 실행한다
// 기존의 for문과 비슷하게 'for((i=1;i<=10;i++))'로 변경
// 해서 사용할수도 있다

// hap에 i 변수의 값을 누적한다
```

# for~in 문

▶ 예제 코드(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
```

▶ 확인

```
 sh for2.sh
“------andor.sh------”
#!/bin/sh
echo "보고 싶은 파일명을 입력하세요.."
read fname
“------case1.sh------”
#!/bin/sh
case "$1" in
start)
“------case2.sh------”
#!/bin/sh
echo "리눅스가 재미있나요?(yes/no)"
read answer
.....
```

# for~in 문

▶ 예제 코드(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
```

▶ 소스 설명

```
// fname 변수에 'ls *.sh'의 실행결과를 하나씩
// 넣어서 4~5행을 반복한다.
// 즉, 파일개수만큼 실행을 반복한다

// 파일이름을 출력한다
// 파일의 앞 3줄을 출력한다
```

# While 문

# While문

## ▶ While 문

While 문은 조건식이 참인 동안에 계속 반복하는 특성을 갖는다

## ▶ 예제 코드(while1.sh)

```
1 #!/bin/sh
2 while [ 1 ]
3 do
4    echo “CentOS 7”
6 done
7 exit 0
```

## ▶ 확인

```
[root@localhost test]# sh while1.sh |more
CentOS7
CentOS7
CentOS7
CentOS7
CentOS7
CentOS7
CentOS7
CentOS7
--More--
```

# While문

▶ 예제 코드(while1.sh)

```
1 #!/bin/sh
2 while [ 1 ]



3 do
4     echo “CentOS 7”
6 done
7 exit 0
```

▶ 소스 설명

```
// 조건식 위치에 [ 1 ] 또는 [ : ] 가 오면 항상 참
// 그러므로 4행을 무한히 반복 실행
// 취소하려면 Ctrl + c 를 누른다
```

# While문

▶ 예제 코드(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 $i + 1'
 8 done
 9 echo "1부터 10 까지의 합:"$hap
10 exit 0
```

▶확인

```
sh while2.sh

1부터 10 까지의 합:55
```

# While문

▶ 예제 코드(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 $i + 1'
8 done
9 echo "1부터 10 까지의 합:"$hap
10 exit 0
```

▶소스 설명

```
// 누적할 hap 변수를 초기화 한다
// 1-10 까지 증가할 i 변수를 선언한다
// i가 10보다 작거나 같으면 6~7행을 실행

// hap에 i의 값을 누적해 저장한다
// i변수의 값을 1씩 증가 시킨다
```

# While문

## ▶ While 응용

비밀번호를 입력받고, 비밀번호가 맞을 때까지 계속 입력받는 스크립트 작성

## ▶ 예제 코드(while3.sh)

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

## ▶확인

```
sh while3.sh
비밀번호를 입력하세요
12
틀렸음.다시입력
34
틀렸음.다시입력
1234
 통과~~
```

# While문

▶ 예제 코드(while3.sh)

```
1 #!/bin/sh
2 echo “비밀번호를 입력하세요.”
3 read mypass
4 while [ $mypass != “1234” ]



5 do
6         echo “틀렸음. 다시 입력”
7         read mypass
8 done
9 echo "통과~~”
10 exit 0
```

▶ 소스 설명

// mypass 변수에 값을 입력받는다
// mypass 변수의 값이 ‘1234’ 가 아니면
// 6~7행을 실행한다. 맞으면 while문 종료

// 다시 mypass 변수에 값을 입력받는다

# Until 문

# Until문

## ▶ Until문

while문과 용도가 거의 같지만, until문은 조건식이 참일 때까지(=거짓인 동안) 계속 반복한다
while2.sh을 동일한 용도로 until문으로 바꾸려면 4행을 다음과 같이 바꾸면 된다

## ▶ 예제 코드(until1.sh)

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

# break,continue,exit,return

# break,continue,exit,return

## ▶ break,continue,exit,return

| | |
|---|---|
| break | 주로 반복문을 종료할 때 사용 |
| continue | 반복문의 조건식으로 돌아가게 함 |
| exit | 해당 프로그램을 완전히 종료 |
| return | 함수 안에서 사용될수 있으며 함수를 호출한 곳으로 돌아가게 함 |

# break,continue,exit,return

## ▶ 예제 코드(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)를 완전히 종료함”
13                      exit 1;;
14 esac;
15 done
16 echo “break를 누르면 while을 빠져나와 지금 이 문장이 출력됨.”
17 exit 0
```

# 기 타

# 사용자 정의 함수

## ▶ 사용자 정의 함수

사용자가 직접 함수를 작성하고 호출할 수 있다

### ▷기본 문법

```
함수이름 ( )        -> 함수를 정의
{
        내용
}
함수이름            -> 함수를 호출
```

### ▷예제 코드(func1.sh)

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

### ▷확인

```
sh func1.sh
프로그램을 시작합니다
함수 안에 들어 왔음
프로그램을 종료합니다
```

# 사용자 정의 함수

## ▶ 사용자 정의 함수

사용자가 직접 함수를 작성하고 호출할 수 있다

### ▷예제코드

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

### ▷소스 설명

2~5행 : 함수를 정의한다
단 이부분은 6행에서 호출하기 전에는 실행되지 않는다
여기서 return문은 함수를 호출한 곳으로 돌아가게 한다
하지만 지금 예에서는 return문이 없어도 된다

6행 : 여기서부터 프로그램이 시작된다

7행 : 함수 이름을 사용하면 함수가 호출된다

# 함수의 파라미터 사용

## ▶ 함수의 파라미터 사용

함수의 파라미터, 즉 인자를 사용하려면 함수를 호출할 때 뒤에서 파라미터를 붙여서 호출하며,
함수 안에서는 $1, $2, ... 로 사용한다

▷기본 문법

```
함수이름 (  )                                -> 함수를 정의
{
        $1 , $2... 등을 사용
}
함수이름 파라미터1 파라미터2        -> 함수를 호출
```

▷예제 코드(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 사용

▷확인

```
sh func2.sh

10 더하기 20을 실행합니다
30
```

▷예제 코드(func2.sh)

```
1 #!/bin/sh
2 hap () {
3           echo ‘expr $1 + $2’          // 넘겨받은 파라미터 $1 과 $2를 더한 값을 출력
4 }
5 echo “10 더하기 20을 실행합니다”
6 hap 10 20                              // 호출할때 함수 이름에 넘겨줄 파라미터를
                                         //  공백으로 분리해서 차례로 적어준다
7 exit 0
```

# eval

## ▶ eval

문자열을 명령문으로 인식하고 실행한다

▷예제코드(eval.sh)+소스설명(//)

```
1 #!/bin/sh
2 str="ls -l /home"
3 echo $str              // 문자열 출력
4 eval $str              // 명령어 실행
5 exit 0
```

▷확인

```
 sh eval.sh

ls -l /home
합계 0
drwx------. 3 user1 user1 78  9월  5 05:00 user1
```

# export

## ▶export

외부 변수로 선언한다. 즉, 선언한 변수를 다른 프로그램에서도 사용할 수 있게 한다

### ▷예제코드(exp1.sh , exp2.sh)

[exp1.sh]
```
1 #!/bin/sh
2 echo $var1
3 echo $var2
4 exit 0
```

▷확인

```
sh exp2.sh

외부 변수
```

[exp2.sh]
```
1 #!/bin/sh
2 var1="지역변수"
3 export var2="외부 변수"
4 sh exp1.sh
5 exit 0
```

### ▷소스 설명

exp1.sh 2 ~ 3행
var1과 var2 변수를 출력한다. var2는 exp2.sh에서 외부변수로 선언했다

exp2.sh 2행
var1에 값을 넣는다. 일반 변수(지역 변수) 이므로 현재 프로그램인 exp2.sh 에서만 사용된다. 즉, exp1.sh의 var1과는 우연히 이름만 같을 뿐 다른 변수다

exp2.sh 3행
var2를 외부 변수로 선언하고 값을 넣는다.
외부 프로그램(exp1.sh)에서도 사용 가능하다

exp2.sh 4행
exp1.sh을 실행한다

# printf

## ▶printf

C언어의 printf() 함수와 비슷하게 형식을 지정해서 출력할 수 있다

### ▷예제코드(printf.sh)

```
1 #!/bin/sh
2 var1=100.5
3 var2="Linux is easy!!~~"
4 printf "%5.2f \n\n \t %s \n" "$var1" "$var2"
5 exit
```

### ▷예제코드(printf.sh)

```
sh printf.sh

100.50

        Linux is easy!!~~
```

### ▷소스 설명

3행 공백이 있으므로 ""로 묶어줘야 한다
4행 %5.2f : 총 5자리이며 소수점 아래 2자리까지 출력
\n : 한줄을 넘기는 개행문자
\t : tab 문자
%s : 문자열을 출력

[주의]
$var2의 경우, 값 중간에 공백이 있으므로, 변수 이름을 ""로 묶어줘야 오류가 발생하지 않는다

# set과 $(명령어)

## ▶set과 $(명령어)

리눅스 명령어를 결과로 사용하려면 '$(명령어)' 형식을 사용해야 한다
또, 결과를 파라미터로 사용하고자 할 때는 'set'명령어와 함께 사용한다

### ▷예제코드(set.sh)

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

### ▷확인

```
sh set.sh

오늘 날짜는 2018. 09. 07. (금) 13:00:20 KST 입니다
오늘은 (금) 요일 입니다
```

### ▷소스 설명

| | |
|---|---|
| 2행 | $(date)는 'date'명령어를 실행한 결과를 보여준다 |
| 3행 | $(date)의 결과가 $1,$2,$3.... 등의 파라미터 변수에 저장 |
| 4행 | 4번째 파라미터인 요일이 출력된다 |

# shift

## ▶shift

파라미터 변수를 왼쪽으로 한 단계식 아래로 쉬프트(이동)시킨다
모든 파라미터 변수를 출력하고 싶거나, 특히 10개가 넘는 파라미터 변수에 접근할때 사용한다
단, $0 파라미터 변수는 변경되지 않는다

### ▷예제코드(shift1.sh)

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

### ▷소스 설명

2~5행 : 함수를 정의한다
단 이부분은 6행에서 호출하기 전에는 실행되지 않는다
여기서 return문은 함수를 호출한 곳으로 돌아가게 한다
하지만 지금 예에서는 return문이 없어도 된다

6행 : 여기서부터 프로그램이 시작된다

7행 : 함수 이름을 사용하면 함수가 호출된다

# shift

## ▶shift

파라미터 변수를 왼쪽으로 한 단계식 아래로 쉬프트(이동)시킨다
모든 파라미터 변수를 출력하고 싶거나, 특히 10개가 넘는 파라미터 번수에 접근할때 사용한다
단, $0 파라미터 변수는 변경되지 않는다

### ▷예제코드(shift1.sh)

```
1 #!/bin/sh
2 myFunc () {
3               echo $1 $2 $3 $3 $4 $5 $6 $7 $8 $9 $10 $11
4}
5 myFunc AAA BBB CCC DDD EEE FFF GGG HHH III JJJ KKK
6 exit
```

### ▷소스 설명

3행 전달받은 파라미터 11개를 출력한다.

5행 11개 파라미터로 myFunc 함수를 호출한다

### ▷확인

```
sh shift1.sh

AAA BBB CCC DDD EEE FFF GGG HHH III AAA0 AAA
```

[AAA0 AAA 출력 이유]

예제코드의 $10, $11은 예상했던 결과 (JJJ KKK)와는 다름
이유는 $10은 $1에 0이붙인 것으로 해석
shift 키워드를 사용해서 해결해야함

# shift

## ▷예제코드 shift 사용(shift2.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
```

## ▷확인

```
sh shift2.sh

AAA BBB CCC DDD EEE FFF GGG HHH III JJJ KKK
```

## ▷소스 설명

3행 결과를 누적할 str 변수를 초기화한다

4행 $1 파라미터가 비어있지 않는 동안에 반복 실행한다
처음 $1은 'AAA'고, 한번 반복 실행하면 5,6행에 의해 $1이 'BBB'가 된다

5행 str 변수에 $1을 추가한다

6행 전체 파라미터를 왼쪽으로 쉬프트시킨다
즉, $2->$1, $3->$2, $4->$3, ... 의 형태로 작업이 일어난다

8행 while문이 끝나면 누적한 str 변수를 출력한다

# Thank You!