

셸스크립트프로그래밍

CentOS의 bash 셸

- 기본 셸은 bash(Bourne Again SHell : '배시 셸')
- bash 셸의 특징
 - Alias 기능(명령어 단축 기능)
 - History 기능(위/아래 화살표키)
 - 연산 기능
 - Job Control 기능
 - 자동 이름 완성 기능(탭키)
 - 프롬프트 제어 기능
 - 명령 편집 기능
- 셸의 명령문 처리 방법
 - (프롬프트) 명령어 [옵션...] [인자...]
 - 예) # rm -rf /mydir

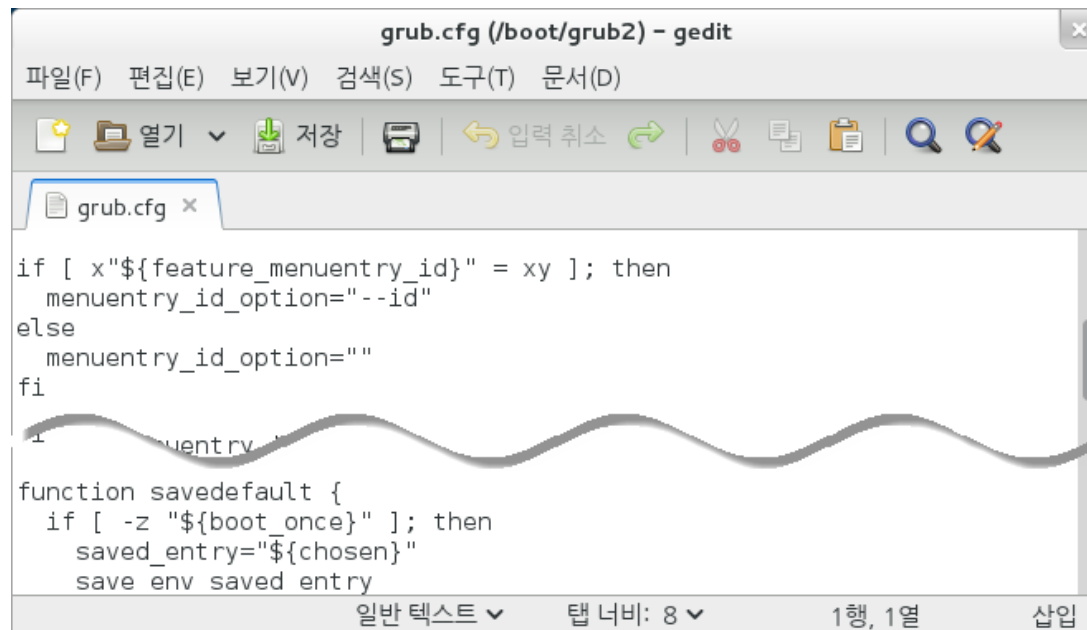
환경 변수

- “echo \$환경변수이름” 으로 확인 가능
- “export 환경변수=값” 으로 환경 변수의 값을 변경
- 주요 환경변수

환경 변수	설명	환경 변수	설명
HOME	현재 사용자의 홈 디렉터리	PATH	실행 파일을 찾는 디렉터리 경로
LANG	기본 지원되는 언어	PWD	사용자의 현재 작업 디렉터리
TERM	로그인 터미널 타입	SHELL	로그인해서 사용하는 셸
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(대개는 `>`)
BASH	bash 셸의 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

셸 스크립트 프로그래밍

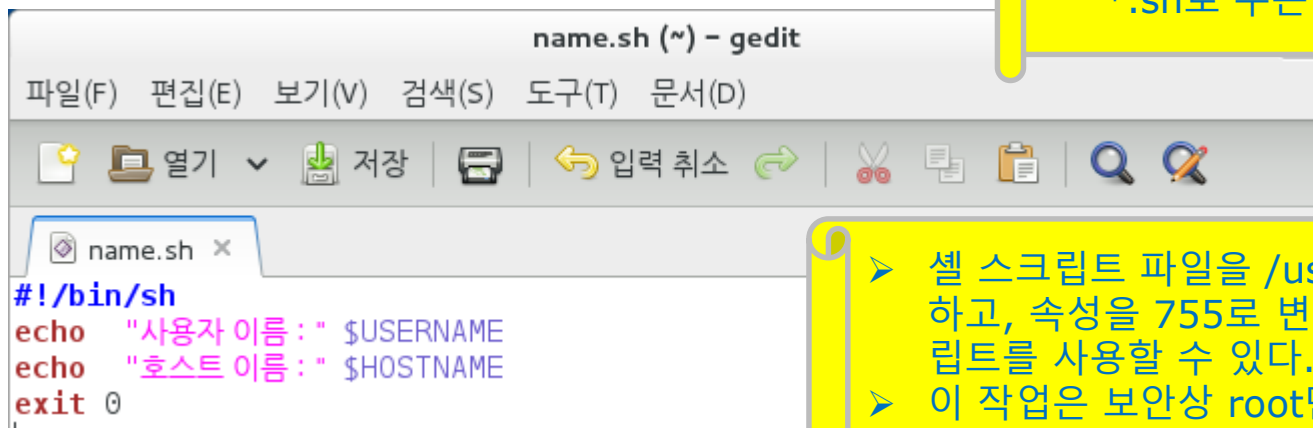
- C언어와 유사하게 프로그래밍이 가능
- 변수, 반복문, 제어문 등의 사용이 가능
- 별도로 컴파일하지 않고 텍스트 파일 형태로 바로 실행
- vi나 gedit으로 작성이 가능
- 리눅스의 많은 부분이 셸 스크립트로 작성되어 있음

A screenshot of a gedit text editor window titled 'grub.cfg (/boot/grub2) - gedit'. The window has a menu bar with '파일(F)', '편집(E)', '보기(V)', '검색(S)', '도구(T)', and '문서(D)'. Below the menu is a toolbar with icons for opening, saving, printing, undo, redo, cut, copy, paste, search, and zoom. The main text area shows the content of 'grub.cfg', which includes conditional logic for menuentry_id_option and a function named 'savedefault'. The status bar at the bottom indicates '일반 텍스트', '탭 너비: 8', '1행, 1열', and '삽입' mode.

```
grub.cfg (/boot/grub2) - gedit
파일(F) 편집(E) 보기(V) 검색(S) 도구(T) 문서(D)
열기 저장 인쇄 입력 취소
grub.cfg x
if [ x"${feature_menuentry_id}" = xy ]; then
  menuentry_id_option="--id"
else
  menuentry_id_option=""
fi
function savedefault {
  if [ -z "${boot_once}" ]; then
    saved_entry="${chosen}"
    save env saved_entry
  fi
}
```

셸 스크립트의 작성과 실행

- vi나 gedit으로 작성



```
name.sh (~) - gedit
파일(F) 편집(E) 보기(V) 검색(S) 도구(T) 문서(D)
열기 저장 입력 취소
name.sh x
#!/bin/sh
echo "사용자 이름:" $USERNAME
echo "호스트 이름:" $HOSTNAME
exit 0
```

➤ 셸 스크립트 파일의 확장명은 되도록 *.sh로 주는 것이 좋다.

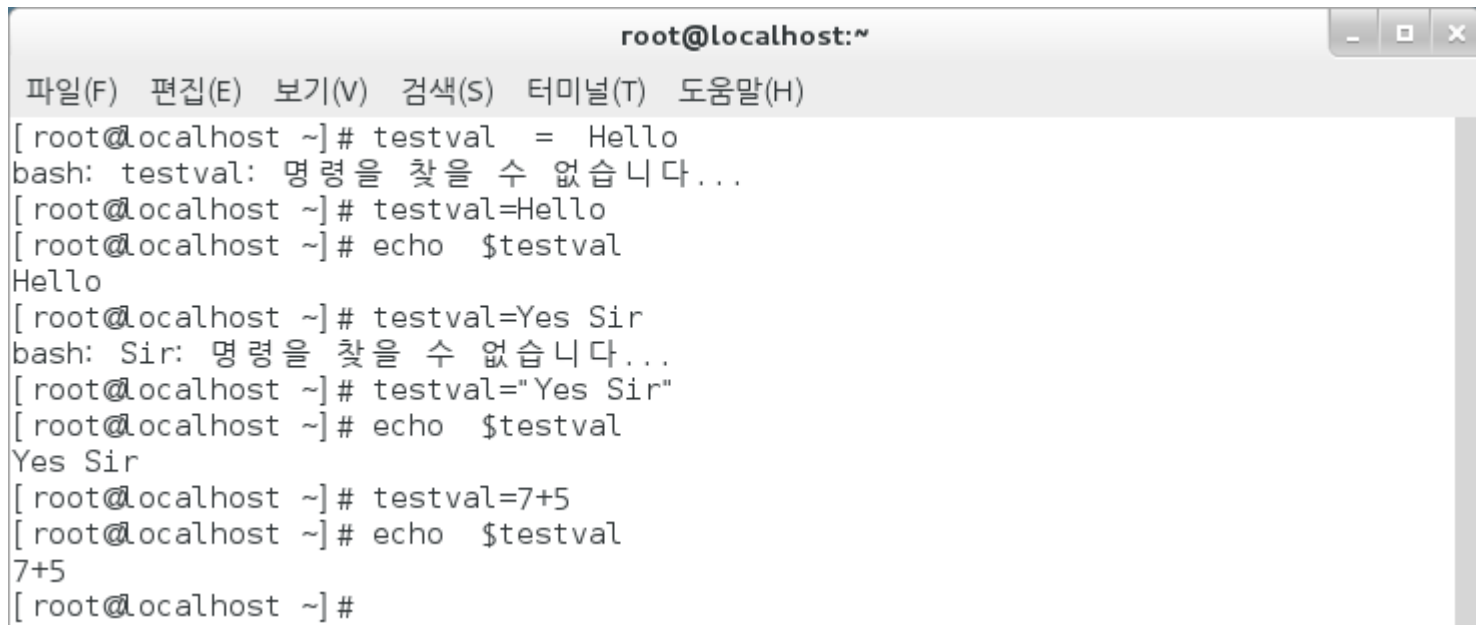
- 셸 스크립트 파일을 /usr/local/bin/ 디렉토리에 복사하고, 속성을 755로 변경해 주면 모든 사용자가 스크립트를 사용할 수 있다.
- 이 작업은 보안상 root만 수행함

- 실행 방법

- ① "sh <스크립트 파일>"로 실행
- ② "chmod +x <스크립트 파일>" 명령으로 실행 가능 속성으로 변경한 후에, "./<스크립트파일>"명령으로 실행

변수의 기본

- 변수를 사용하기 전에 미리 선언하지 않으며, 변수에 처음 값이 할당되면서 자동으로 변수가 생성
- 모든 변수는 '문자열(String)'로 취급
- 변수 이름은 대소문자를 구분
- 변수를 대입할 때 '='좌우에는 공백이 없어야 함



```
root@localhost:~  
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)  
[root@localhost ~]# testval = Hello  
bash: testval: 명령을 찾을 수 없습니다...  
[root@localhost ~]# testval=Hello  
[root@localhost ~]# echo $testval  
Hello  
[root@localhost ~]# testval=Yes Sir  
bash: Sir: 명령을 찾을 수 없습니다...  
[root@localhost ~]# testval="Yes Sir"  
[root@localhost ~]# echo $testval  
Yes Sir  
[root@localhost ~]# testval=7+5  
[root@localhost ~]# echo $testval  
7+5  
[root@localhost ~]#
```

변수의 입력과 출력

- '\$' 문자가 들어간 글자를 출력하려면 ' '로 묶어주거나 앞에 '\ '를 붙임.
- " "로 변수를 묶어줘도 된다.

01 #!/bin/sh

02 myvar="Hi Woo"

03 echo \$myvar

04 echo "\$myvar"

05 echo '\$myvar'

06 echo \ \$myvar

07 echo 값 입력 :

08 read myvar

09 echo '\$myvar' = \$myvar

10 exit 0

```
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T)
[root@localhost ~]# sh var1.sh
Hi Woo
Hi Woo
$myvar
$myvar
값 입력 :
안녕하세요?
$myvar = 안녕하세요?
[root@localhost ~]#
```

숫자 계산

- 변수에 대입된 값은 모두 문자열로 취급
- 변수에 들어 있는 값을 숫자로 해서 +, -, *, / 등의 연산을 하려면 expr을 사용
- 수식에 괄호 또는 곱하기(*)는 그 앞에 꼭 역슬래시(\) 붙임

01 #!/bin/sh

02 num1=100

03 num2=\$num1+200

04 echo \$num2

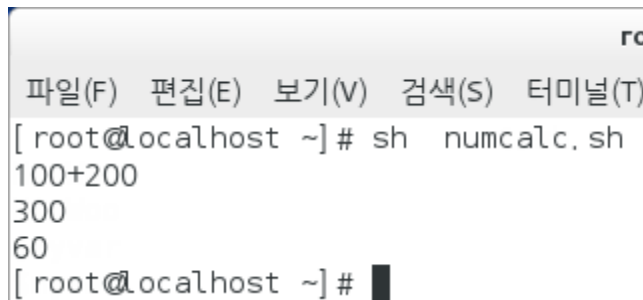
05 num3='expr \$num1 + 200'

06 echo \$num3

07 num4='expr \(\$num1 + 200\) / 10 * 2'

08 echo \$num4'

09 exit 0



```
ro
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T)
[root@localhost ~]# sh numcalc.sh
100+200
300
60
[root@localhost ~]#
```


파라미터(Parameter) 변수

- 파라미터 변수는 \$0, \$1, \$2...의 형태를 가짐
- 전체 파라미터는 \$*로 표현

예)

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

01 #!/bin/sh

02 echo "실행파일 이름은 <\$0>이다"

03 echo "첫번째 파라미터는 <\$1>이고, 두번째 파라미터는 <\$2>다"

04 echo "전체 파라미터는 <\$*>다"

05 exit 0

```
root@localhost:~  
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)  
[root@localhost ~]# sh paravar.sh 값1 값2 값3  
실행파일 이름은 <paravar.sh> 이다  
첫번째 파라미터는 <값1> 이고, 두번째 파라미터는 <값2> 다  
전체 파라미터는 <값1 값2 값3> 다  
[root@localhost ~]#
```

기본 if 문

- 형식

if [조건]

then

 참일 경우 실행

fi

01 #!/bin/sh

02 if ["woo" = "woo"]

03 then

04 echo "참입니다"

05 fi

06 exit 0

➤ "[조건]"의 사이의 각 단어에는 모두 공백이 있어야 한다

```
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T)
[root@localhost ~]# sh if1.sh
참입니다
[root@localhost ~]#
```

if~else 문

- 형식

if [조건]

then

참일 경우 실행

else

거짓인 경우 실행

fi

01 #!/bin/sh

02 if ["woo" != "woo"]

03 then

04 echo "참입니다"

05 else

06 echo "거짓입니다"

07 fi

08 exit 0

➤ 중복 if 문을 위해서 else if가 합쳐진 elif문도 사용할 수 있다.

```
root
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T)
[root@localhost ~]# sh if2.sh
거짓입니다
[root@localhost ~]#
```

조건문에 들어가는 비교 연산자

문자열 비교	결과
"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

```
01 #!/bin/sh
02 if [ 100 -eq 200 ]
03 then
04     echo "100과 200은 같다."
05 else
06     echo "100과 200은 다르다."
07 fi
08 exit 0
```

산술 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

```
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T)
[ root@localhost ~]# sh if3.sh
100과 200은 다르다.
[ root@localhost ~]#
```

파일과 관련된 조건

```
01 #!/bin/sh
02 fname=/lib/systemd
    /system/httpd.service
03 if [ -f $fname ]
04 then
05     head -5 $fname
06 else
07     echo "웹 서버가 설치되지 않았습니다."
08 fi
09 exit 0
```

파일 조건	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

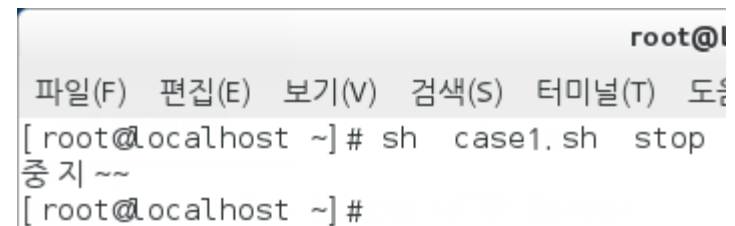
```
root@localhost:~
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
[root@localhost ~]# sh if4.sh
[Unit]
Description=The Apache HTTP Server
After=network.target remote-fs.target nss-lookup.target

[Service]
[root@localhost ~]# █
```

case~esac 문

- if 문은 참과 거짓의 두 경우만 사용 (2중분기)
- 여러 가지 경우의 수가 있다면 case 문 (다중분기)

```
01 #!/bin/sh
02 case "$1" in
03     start)
04         echo "시작~~" ;;
05     stop)
06         echo "중지~~" ;;
07     restart)
08         echo "다시 시작~~" ;;
09     *)
10         echo "뭔지 모름~~" ;;
11 esac
12 exit 0
```

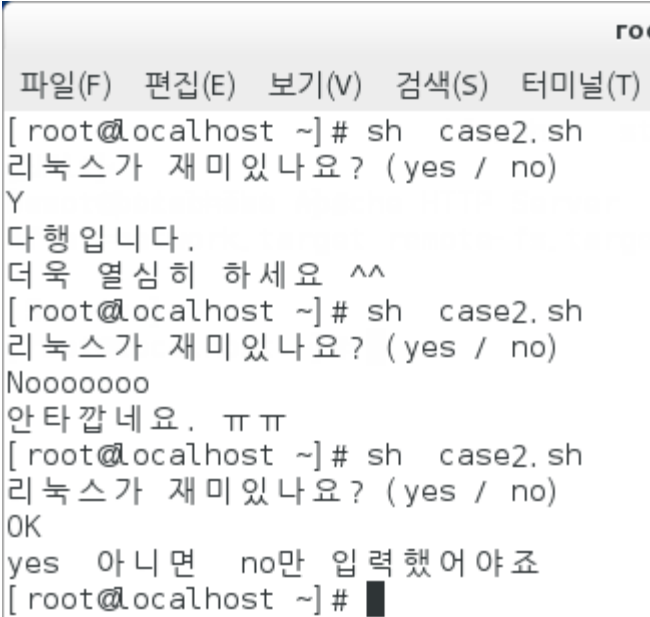


A terminal window screenshot showing the execution of a shell script. The prompt is 'root@localhost ~'. The user enters 'sh case1.sh stop'. The script outputs '중지~~' (Jungji~~). The prompt returns to 'root@localhost ~'.

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

case~esac 문

```
01 #!/bin/sh
02 echo "리눅스가 재미있나요? (yes / no)"
03 read answer
04 case $answer in
05     yes | y | Y | Yes | YES)
06         echo "다행입니다."
07         echo "더욱 열심히 하세요 ^^" ;;
08     [nN]*)
09         echo "안타깝네요. □ □ " ;;
10     *)
11         echo "yes 아니면 no만
                                입력했어야죠"
12         exit 1 ;;
13 esac
14 exit 0
```



```
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T)
[ root@localhost ~]# sh case2.sh
리눅스가 재미있나요? (yes / no)
Y
다행입니다.
더욱 열심히 하세요 ^^
[ root@localhost ~]# sh case2.sh
리눅스가 재미있나요? (yes / no)
Nooooooooo
안타깝네요. □ □
[ root@localhost ~]# sh case2.sh
리눅스가 재미있나요? (yes / no)
OK
yes 아니면 no만 입력했어야죠
[ root@localhost ~]#
```

AND, OR 관계 연산자

- and는 '-a' 또는 '&&'를 사용
- or는 '-o' 또는 '||'를 사용

```
01 #!/bin/sh
02 echo "보고 싶은 파일명을 입력하세요."
03 read fname
04 if [ -f $fname ] && [ -s $fname ] ; then
05     head -5 $fname
06 else
07     echo "파일이 없거나,
           크기가 0입니다."
08 fi
09 exit 0
```

```
root@localhost:~  
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)  
[root@localhost ~]# sh andor.sh  
보고 싶은 파일명을 입력하세요.  
/lib/systemd/system/nofile.service  
파일이 없거나, 크기가 0입니다.  
[root@localhost ~]# sh andor.sh  
보고 싶은 파일명을 입력하세요.  
/lib/systemd/system/httpd.service  
[Unit]  
Description=The Apache HTTP Server  
After=network.target remote-fs.target nss-lookup.target  
[Service]  
[root@localhost ~]#
```


반복문 – for문

- 형식

for 변수 in 값1 값2 값3 ...

do

반복할 문장

done

➤ 3행은
for((i=1;i<=10;i++)) 또는 for i in 'seq 1 10'
로 변경 할 수 있음

01 #!/bin/sh

02 hap=0

03 for i in 1 2 3 4 5 6 7 8 9 10

04 do

05 hap='expr \$hap + \$i'

06 done

07 echo "1부터 10까지의 합: "\$hap

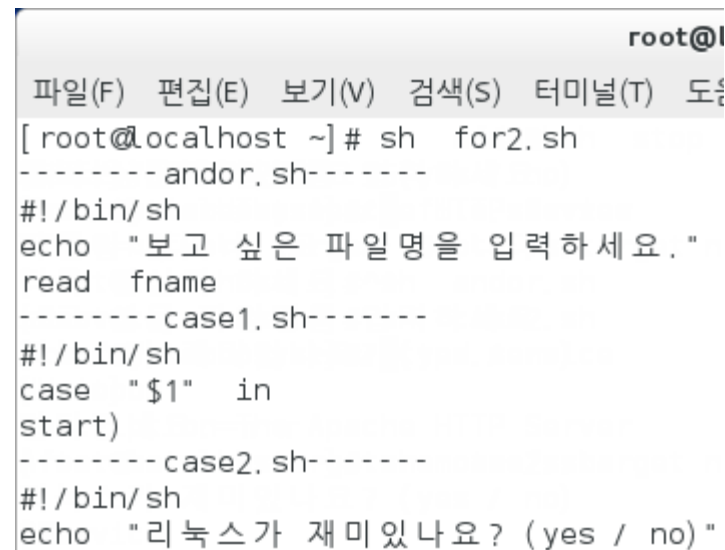
08 exit 0

```
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T)
[root@localhost ~]# sh for1.sh
1부터 10까지의 합: 55
[root@localhost ~]#
```

반복문 – for문 (2)

- 현재 디렉터리에 있는 셸 스크립트 파일(*.sh)의 파일명과 앞 3줄을 출력하는 프로그램

```
01 #!/bin/sh
02 for fname in $(ls *.sh)
03 do
04     echo "-----$fname-----"
05     head -3 $fname
06 done
07 exit 0
```



```
root@l
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도
[root@localhost ~]# 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)"
```

반복문 – while문 (1)

- 조건식이 참인 동안에 계속 반복

```
01 #!/bin/sh
02 while [ 1 ]
03 do
04   echo "CentOS 7"
05 done
06 exit 0
```

➤ [1] 또는 [:]가 오면 항상 참이 됨.
그러므로 4행을 무한 루프로 반복함.

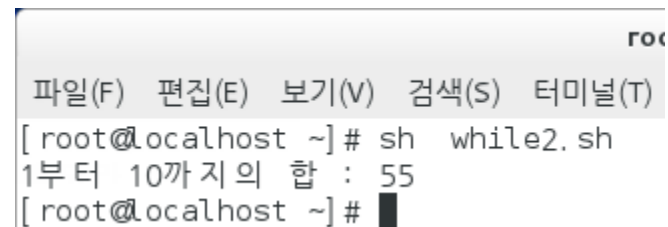
```
root@lo
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움
[root@localhost ~]# sh while1.sh
CentOS 7
CentOS 7
CentOS 7
CentOS 7
CentOS 7
CentOS 7
```

반복문 – while문 (2)

- 1에서 10까지의 합계를 출력 ('반복문 – for문 (1)' 슬라이드와 동일)

```
01 #!/bin/sh
02 hap=0
03 i=1
04 while [ $i -le 10 ]
05 do
06   hap='expr $hap + $i'
07   i='expr $i + 1'
08 done
09 echo "1부터 10까지의 합 : "$hap
10 exit 0
```

- until 문은 조건식이 참일 때까지(= 거짓인 동안) 계속 반복
- 4행을 until 문으로 바꾸면, until [\$i -gt 10]

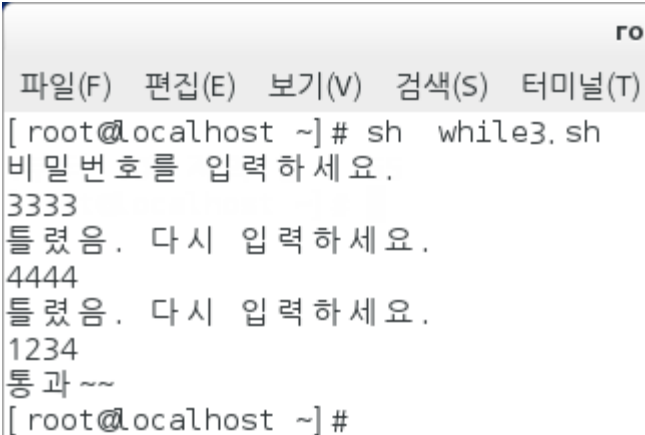


A terminal window titled 'root' with a menu bar containing '파일(F)', '편집(E)', '보기(V)', '검색(S)', and '터미널(T)'. The terminal shows the command `[root@localhost ~]# sh while2.sh` being executed, followed by the output `1부터 10까지의 합 : 55`. The prompt `[root@localhost ~]#` is visible at the bottom.

반복문 – while문 (3)

- 비밀번호를 입력받고, 비밀번호가 맞을 때까지 계속 입력받는 스크립트

```
01 #!/bin/sh
02 echo "비밀번호를 입력하세요."
03 read mypass
04 while [ $mypass != "1234" ]
05 do
06     echo "틀렸음. 다시 입력하세요."
07     read mypass
08 done
09 echo "통과~~"
10 exit 0
```



A terminal window titled 'ro' with a menu bar containing '파일(F)', '편집(E)', '보기(V)', '검색(S)', and '터미널(T)'. The terminal shows the execution of a shell script named 'while3.sh'. The prompt is '[root@localhost ~]# sh while3.sh'. The script outputs '비밀번호를 입력하세요.' and prompts for input. The user enters '3333', and the script outputs '틀렸음. 다시 입력하세요.'. The user enters '4444', and the script outputs '틀렸음. 다시 입력하세요.'. The user enters '1234', and the script outputs '통과~~'. The prompt returns to '[root@localhost ~]#'.

```
ro
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T)
[root@localhost ~]# sh while3.sh
비밀번호를 입력하세요.
3333
틀렸음. 다시 입력하세요.
4444
틀렸음. 다시 입력하세요.
1234
통과~~
[root@localhost ~]#
```

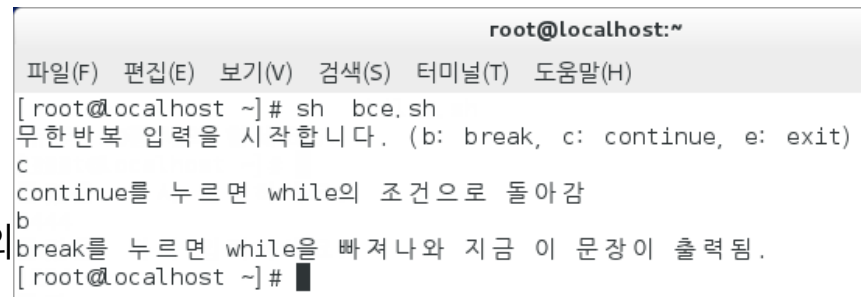
until 문

- while문과 용도가 거의 같지만, until문은 조건식이 참일 때까지(=거짓인 동안) 계속 반복한다.
- while2.sh를 동일한 용도로 until문으로 바꾸려면 4행을 다음과 같이 바꾸면 된다.
 - `until [$i -gt 10]`

break, continue, exit, return 문

- break는 주로 반복문을 종료할 때 사용되며, continue는 반복문의 조건식으로 돌아가게 함. exit는 해당 프로그램을 완전히 종료함. Return은 함수 안에서 사용될 수 있으며 함수를 호출한 곳으로 돌아가게 함.

```
01 #!/bin/sh
02 echo "무한반복 입력을 시작합니다(b: break, c: continue, e: exit)"
03 while [ 1 ]; do
04 read input
05 case $input in
06     b | B )
07         break ;;
08     c | C )
09         echo "continue를 누르면 while의
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
```



```
root@localhost:~
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
[root@localhost ~]# 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@localhost ~]#
```

사용자 정의 함수

- 형식

함수이름 () { → 함수를 정의
내용들...

}

함수이름 → 함수를 호출

01 #!/bin/sh

02 myFunction () {

03 echo "함수 안으로 들어 왔음"

04 return

05 }

06 echo "프로그램을 시작합니다."

07 myFunction

08 echo "프로그램을 종료합니다."

09 exit 0

```
ro
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T)
[root@localhost ~]# sh func1.sh
프로그램을 시작합니다.
함수 안으로 들어 왔음
프로그램을 종료합니다.
[root@localhost ~]#
```


함수의 파라미터 사용

- 형식

함수이름 () { → 함수를 정의

\$1, \$2 ... 등을 사용

}

함수이름 파라미터1 파라미터2 ... → 함수를 호출

01 #!/bin/sh

02 hap () {

03 echo 'expr \$1 + \$2'

04 }

05 echo "10 더하기 20을 실행합니다"

06 hap 10 20

07 exit 0

```
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T)
[ root@localhost ~]# sh func2.sh
10 더하기 20을 실행합니다
30
[ root@localhost ~]#
```

eval

- 문자열을 명령문으로 인식하고 실행

01 #!/bin/sh

02 str="ls -l anaconda-ks.cfg"

03 echo \$str

04 eval \$str

05 exit 0

```
root@localhost:~  
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)  
[root@localhost ~]# sh eval.sh  
ls -l anaconda-ks.cfg  
-rw-----, 1 root root 1557  8월  8 10:31 anaconda-ks.cfg  
[root@localhost ~]#
```

export

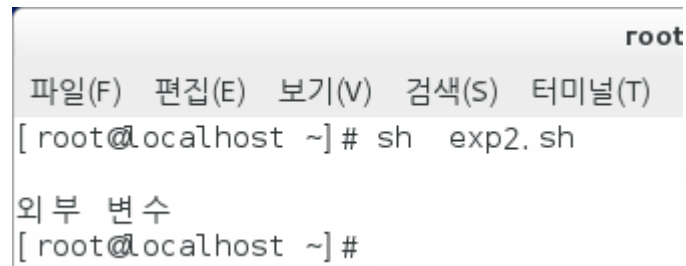
- 외부 변수로 선언해 준다. 즉, 선언한 변수를 다른 프로그램에서도 사용할 수 있도록 해줌

- exp1.sh

```
01 #!/bin/sh
02 echo $var1
03 echo $var2
04 exit 0
```

- exp2.sh

```
01 #!/bin/sh
02 var1="지역 변수"
03 export var2="외부 변수"
04 sh exp1.sh
05 exit 0
```



```
root
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T)
[root@localhost ~]# sh exp2.sh
외부 변수
[root@localhost ~]#
```

printf

- C언어의 printf() 함수와 비슷하게 형식을 지정해서 출력

01 #!/bin/sh

02 var1=100.5

03 var2="재미있는 리눅스~~"

04 printf "%5.2f \n\n \t %s \n" \$var1 "\$var2"

05 exit

```
root
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T)
[root@localhost ~]# sh printf.sh
100.50

      재미있는 리눅스 ~~
[root@localhost ~]# █
```

set과 \$(명령어)

- 리눅스 명령어를 결과로 사용하기 위해서는 \$(명령어) 형식을 사용
- 결과를 파라미터로 사용하고자 할 때는 set과 함께 사용

01 #!/bin/sh

02 echo "오늘 날짜는 \$(date) 입니다."

03 set \$(date)

04 echo "오늘은 \$4 요일 입니다."

05 exit 0

```
root@localhost:~  
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)  
[root@localhost ~]# sh set.sh  
오늘 날짜는 2013. 10. 12. (토) 17: 25: 40 KST 입니다.  
오늘은 (토) 요일 입니다.  
[root@localhost ~]#
```

shift (1)

- 파라미터 변수를 왼쪽으로 한 단계씩 아래로 쉬프트시킴
- 10개가 넘는 파라미터 변수에 접근할 때 사용
- 단, \$0 파라미터 변수는 변경되지 않음

- 원하지 않는 결과의 소스

01 #!/bin/sh

02 myfunc () {

03 echo \$1 \$2 \$3 \$4 \$5 \$6 \$7 \$8 \$9 \$10 \$11

04 }

05 myfunc AAA BBB CCC DDD EEE FFF GGG HHH III JJJ KKK

06 exit 0

```
root@localhost:~  
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)  
[root@localhost ~]# sh shift1.sh  
AAA BBB CCC DDD EEE FFF GGG HHH III AAA0 AAA1  
[root@localhost ~]#
```

shift (2)

- shift 사용을 통한 앞 문제점을 해결

```
01 #!/bin/sh
02 myfunc() {
03     str=""
04     while [ "$1" != "" ]; do
05         str="$str $1"
06         shift
07     done
08     echo $str
09 }
10 myfunc AAA BBB CCC DDD EEE FFF GGG HHH III JJJ KKK
11 exit 0
```

```
root@localhost:~
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
[root@localhost ~]# sh shift2.sh
AAA BBB CCC DDD EEE FFF GGG HHH III JJJ KKK
[root@localhost ~]# █
```