LINUX

환경변수와
셸스크립트

환경 변수

- 리눅스 시스템 위에서 동작하는 프로그램들이 프로그램 외부에서 설정한 정보를 참조할 수 있도록 설정할 수 있는 변수
- 쉘이 관리

환경 변수 값 변경: #export 환경변수=값
환경 변수 확인: #printenv

환경변수명	설명	환경변수명	설명
HOME	사용자의 홈 디렉터리 경로	PATH	실행 파일을 찾는 디렉터리 경로
LANG	기본 지원 언어	PWD	사용자의 현재 작업 디렉터리
TERM	로그인 터미널 타입	SHELL	사용하고 있는 쉘
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 명령의 확장자 색상 옵션
MAIL	메일을 보관하는 경로	OSTYPE	운영체제 타입

사용자 프로파일(Profile)과 환경 변수

- 사용자가 로그인하게 되면 사용자 프로파일이 읽어지고 나서 쉘 프롬포트가 나타남
- 사용자 프로파일에 의해 사용자 환경이 초기화되고 변수들이 새롭게 적용됨
- 사용자의 환경 변수(전역 변수): /etc/profile, /etc/bashrc
- 사용자의 지역 변수: ~/.profile, ~/.bashrc

etc/profile

- 시스템 사용자 전체에 적용할 셸 환경 변수를 담고 있는 파일

etc/profile.d

- 사용자 로그인 시 /etc/profile에 의해 실행되는 스크립트 파일들이 있는 디렉터리

/etc/bashrc

- 사용자 로그인 시 사용자 계정에 있는 ~/.bashrc 파일에 의해서 실행되는 파일
- 사용자에게 적용할 alias와 셸 환경 값을 설정하고 있는 파일

~/.profile

- 사용자의 ~/.profile에서는 ~/.bashrc를 읽어서 실행되도록 함
- 실행 경로 환경 변수를 적용해 줄 수 있음
- PATH=$PATH:추가할경로:추가할경로
- 설정한 경로가 적용되기 위해서는 재로그인 혹은 #export PATH 실행

~/.bashrc

- 해당 사용자에 대한 profile 설정이 있는 파일
- 사용자가 사용하는 alias설정 등

bash셸

- 셸: 사용자가 입력한 명령을 해석해 커널로 전달하거나 커널의 처리 결과를 사용자에게 전달함
- bashShell: sh(Borune shell)을 기반으로 Korn Shell(ksh)과 C Shell(csh)을 합친 것

Alias 기능 (명령 단축 기능)

History 기능

연산 기능

자동 완성 기능

명령 편집 기능

명령에 별칭을 지정하여 해당 명령을 별칭으로 사용하는 기능
셸 세션이 생존하는 동안에만 지속됨
~/.bashrc 또는 /etc/bashrc 또는 /etc/bash.bashrc 에
직접기록하거나 source로 활성화 후 사용

alias 별칭='명령어'
unalias 별칭 or unalias *

단축키	기능
[Ctrl + a] / Home	커서를 맨 왼쪽으로 이동
[Ctrl + e] / End	커서를 맨 오른쪽으로 이동
[Ctrl + u]	명령행 전체 삭제
[Ctrl + y]	삭제 취소

history 환경변수	기능
HISTSIZE	저장하는 명령어 개수 지정
HISTFILESIZE	실제 히스토리 파일의 크기
HISTFILE	히스토리 파일의 위치
HISTCONTROL	중복되는 명령의 기록유무 지정 =erasedups : 히스토리에서 중복제거 =ignoredups : 이전 행과 일치하면 등록안함 =ignorespace : 공백은 기록하지 않는다.

출처: https://wiseworld.tistory.com/47

셸 스크립트

- 셸 스크립트: C언어와 유사한 방법으로 실행시킬 수 있는 스크립트 파일을 작성하는 것
- 별도로 컴파일하지 않고 텍스트 파일 형태로 셸에서 바로 실행 가능
- 확장자를 별도로 요구하지 않지만 일반적으로 .sh를 사용하고, 첫 줄은 사용하는 shell의 이름을 주석(#)으로 작성

셸 스크립트 작성 및 실행

```
  GNU nano 4.8                                  name.sh
#!/bin/sh
echo "사용자 이름: " $USER
echo "홈 디렉터리: " $HOME
exit 0 #종료코드 반환(0: 성공, 1~255: 오류 코드)
```

```
root@Server:~/바탕화면# sh name.sh
사용자 이름:  root
홈 디렉터리:  /root
root@Server:~/바탕화면# ./name.sh
사용자 이름:  root
홈 디렉터리:  /root
```

변수

- 변수를 사용하기 전에 미리 선언하지 않으며, 처음 변수에 값이 할당되면 자동으로 변수가 생성됨
- **변수에 넣는 모든 값은 문자열(String)으로 취급**
- **변수 이름은 대소문자를 구별**
- **변수를 대입할 때 '=' 좌우에는 공백이 없어야 함**
- **$가 들어간 문자를 출력할 때는 ''로 묶거나, $앞에 ₩를 붙여서 사용**
- **""로 변수를 묶어도 되고 묶지 않아도 됨**

```
root@Server:~/바탕화면# testVar = Hello
testVar: 명령을 찾을 수 없습니다
root@Server:~/바탕화면#
root@Server:~/바탕화면# testVar=hello
root@Server:~/바탕화면# echo $testVar
hello
root@Server:~/바탕화면# testVar=Now test
root@Server:~/바탕화면# echo $testVar
hello
root@Server:~/바탕화면# testVar="Now test"
root@Server:~/바탕화면# echo $testVar
Now test
```

```
root@Server:~/바탕화면# testVar=3+4
root@Server:~/바탕화면# echo $testVar
3+4
root@Server:~/바탕화면# echo testVar
testVar
root@Server:~/바탕화면# testVar="3+4"
root@Server:~/바탕화면# echo $testVar
3+4
```

```
#!/bin/sh
myVar="HI!"
echo $myVar
echo $"myVar"
echo "$myVar"
echo '$myVar'
echo \$myVar
echo 값 입력:
read myVar
echo '$myVar' = $myVar
exit 0
```

```
root@Server:~/바탕화면# sh var1.sh
HI!
$myVar
HI!
$myVar
$myVar
값 입력:
123
$myVar = 123
```

파라미터(Parameter)와 인자(Argument)

- 파라미터: $0, $1 등의 형태를 가지며, 명령어에 포함된 모든 단어를 $변수 형식으로 지정
- 명령 전체의 파라미터 변수는 $*로 지정
- 인자: 명령어 뒤에 명령에 필요한 추가 정보나 실행 방법을 작성하여, 파라미터로 보내주는 데이터

apt -y install gftp				
명령	apt	-y	install	gftp
파라미터	$0	$1	$2	$3

형태	설명	예제
$0	프로그램 또는 스크립트 이름	run.sh
$1 ~ $9	프로그램 뒤에 따라오는 인자값	$1=aa, $2=bb, $3=cc, $4=dd
$*	모든 인자 $argv[*], $argv와 동일	aa bb cc dd
$$	현재 실행중인 프로그램의 PID	13287
$!	백그라운드 모드로 실행된 마지막 작업의 PID	
$@	$*와 동일	aa bb cc dd
$-	현재의 셸 옵션	
$?	마지막으로 실행된 명령의 종료상태	0

연산

- 변수에 들어 있는 값에 사칙연산을 적용시키기 위해서는 expr 키워드와 `(역따옴표, 백팁)을 사용하거나, $(())사용
  `expr 산술식 ` 혹은 $((산술식))
- 수식에 괄호를 사용하기 위해서는 '₩'필수. * 사용 시 '₩' 필수

파라미터 사용

```
 GNU nano 4.8                  parameter.sh
#!/bin/sh
echo "실행파일 이름은 <$0>이다"
echo "첫번째 파라미터는 <$1>이고, 두번째 파라미터는 <$2>다"
echo "전체 파라미터는 <$*>다"
exit 0
```

```
root@Server:~/바탕화면# sh parameter.sh ls -l /etc
실행파일 이름은 <parameter.sh>이다
첫번째 파라미터는 <ls>이고, 두번째 파라미터는 <-l>다
전체 파라미터는 <ls -l /etc>다
```

연산

```
echo `expr 100 \* 200`
echo $((100 * 200))
```

```
echo "$1+$2=$(($1 + $2))입니다"
exit 0
```

```
 GNU nano 4.8                   numclac.sh
#!bin/sh
num1=100
#문자로 더해짐
num2=$num1+200
echo $num2
#숫자계산
num3=`expr $num1 + 200`
echo $num3
#단어마다 띄어쓰기 필수
num4=`expr \( $num1 + 200 \) / 10 \* 2`
echo $num4
exit 0
```

조건문1: if

- if - elif - else 로 작성가능하며, ['조건'] 작성시 띄어쓰기 필수

```
if [ 조건1 ] then;
        조건1이 참일경우 실행
elif [ 조건2 ]
then
        조건2가 참일경우 실행
else
        조건이 전부 만족하지 않을 경우 실행
fi
```

문자열 비교	결과
문자열1 = 문자열2	두 문자열이 같으면 참
문자열1 != 문자열2	두 문자열이 같지 않으면 참
-n 문자열	문자열이 NULL이 아니면 참
-z 문자열	문자열이 NULL이면 참

```
  GNU nano 4.8                    ifVar.sh
#!/bin/sh
if [ test = $1 ]; then
        echo "Answer: test"
elif [ final = $1 ]; then
        echo "Answer: final"
else
        echo "Answer: XXX"
fi

exit 0
```

조건문1: if

산술 비교	결과
수식1 -eq 수식2	두 수식이 같으면 참
수식1 -nq 수식2	두 수식이 같지 않으면 참
수식1 -gt 수식2	수식1이 크면 참
수식1 -ge 수식2	수식1이 크거나 같으면 참
수식 -lt 수식2	수식1이 작으면 참
수식 -le 수식2	수식1이 작거나 같으면 참
!수식	수식이 거짓이면 참

파일 조건	결과
-d 파일명	파일이 디렉터리면 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 가능 상태면 참
-x 파일명	파일이 실행 가능 상태면 참

GNU nano 4.8 ifVar2.sh

```
#!/bin/sh
if [ 100 -eq 200 ]; then
        echo "100과 200은 같다"
elif [ 100 -lt 200 ]; then
        echo "100은 200보다 작다"
fi

exit 0
```

GNU nano 4.8 ifVar3.sh

```
#!/bin/bash
fname=$1
if [ -f $fname ]; then
        tail -3 $fname
else
        echo "해당 파일은 디렉터리입니다."
fi
exit 0
```

조건문2: case

- 조건에 해당하지 않는 default 값은 *로 표시
- AND: && 또는 -a를 사용
- OR: || 또는 -o를 사용

```
case 비교할데이터 in
        값1)
                값1과 일치할 때 실행문;;
        값2)
                값2와 일치할 때 실행문;;
        *)
                어떤것도 일치하지 않을 때 실행문;;
esac
```

GNU nano 4.8 case1

```
#!/bin/sh
case "$1" in
    start)
        echo "시작";;
    stop)
        echo "중지";;
    restart)
        echo "다시시작";;
    *)
        echo "뭔지모름";;
esac
exit 0
```

GNU nano 4.8 case2.sh

```
#!/bin/sh
echo "리눅스가 재미있나요? (yes or no)"
read answer
case $answer in
    yes | y | Y | Yes | YES)
        echo 다행이네요. 파이팅!;;
    [nN]*) #정규식사용
        echo 아이고..ㅠㅠ조금만 더 힘내요!;;
    *)
        echo yes or no만 입력가능합니다만..?
        exit 1;;
esac
exit 0
```

반복문: for와 while

- for: for - in 으로 작성가능하며 변수에 각각 값을 넣은 후 do 안에 있는 '반복할 문장'을 실행
- while: 조건식이 참인 동안 계속 '반복할 문장' 실행
- until: 조건식이 거짓인 동안 계속 '반복할 문장' 실행

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

```
while [ 조건식 ]
        do
                반복할 문장
        done
```

```
until [ 조건식 ]
        do
                반복할 문장
        done
```

GNU nano 4.8 forin.sh

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

GNU nano 4.8 forin.sh

```
#!/bin/sh
hap=0
for i in $(seq 5 10)
    do
        hap=`expr $hap + $i`
    done
echo "5부터 10까지의 합: "$hap
exit 0
```

```
#!/bin/sh
for fname in $(ls *.sh)
    do
        echo --------$fname----------
        head -3 $fname
    done
exit 0
```

```
#!/bin/sh
hap=0
i=1
while [ $i -le 10 ]
do
    hap=`expr $hap + $i`
    i=`expr $i + 1`
done
echo "1부터 10까지의 합: $hap"
exit 0
```

```
#!/bin/sh
echo "비밀번호를 입력하세요."
read mypass
while [ $mypass != "1234" ]
do
    echo "틀렸음. 다시 입력하세요"
    read mypass
done
echo "통과!"
exit 0
```

break, continue, exit, return

- break: 반복문을 즉시 종료
- continue: 반복문의 조건식으로 돌아감
- exit: 프로그램을 완전히 종료
- return: 함수를 호출한 곳으로 돌아감

사용자 정의 함수

```
함수명(){ #함수정의
        내용
}
함수명 #호출
```

```
#!/bin/sh
myFunction(){
    echo "함수 안으로 들어옴"
    return
    echo "이건 실행 될 수 없음"
}
echo Program start
myFunction
echo Program end
exit 0
```

```
#!/bin/sh
echo 더할 두 숫자를 입력하세요
read num1 num2
myFunction(){
    sum=$(($1 + $2))
    return $sum
}
myFunction $num1 $num2
result=$?
echo 두 숫자의 합은: $result
exit 0
```

이 외

- eval: 문자열을 명령문으로 인식하고 실행
- export: 변수를 외부 변수로 선언하여 다른 프로그램에서도 사용할 수 있게 함 (환경변수로 설정)
- set: 명령어의 결과를 파라미터로 재사용하고자 할 때 설정하는 명령어
- print: C언어의 printf()함수와 비슷하게 형식지정자 사용 가능
- shift: 파라미터 변수를 왼쪽으로 한 단계씩 아래로 시프트(이동)시킴

export

```
#!/bin/sh
echo $var1
echo $var2
exit 0
```

```
#!/bin/sh
var1="지역변수"
export var2="외부변수"
sh exp1.sh
exit 0
```

set

```
#!/bin/sh
echo 오늘 날짜는 $(date)입니다.
set $(date)
echo 오늘은 $4요일 입니다.
set $(ls)
echo 현재 디렉터리의 첫번째 파일은 $1입니다.
exit 0
```

shift

```
#!/bin/sh
myfunc() {
    str=""
    while [ "$1" != "" ]; do
        str="$str $1"
        shift
    done
    echo $str
}
myfunc AAA BBB CCC DDD ABC ACD ADD ZZZ
exit 0
```

수고하셨습니다