



LINUX 개발환경(2)

- **LINUX Shell 프로그래밍**
 - 변수
 - 표준 입력
 - 연산
 - 흐름제어(조건문,반복문,패턴검색)
 - 디버깅
 - 함수

LINUX Shell 프로그래밍

□ Shell Script

- 셸이나 명령 행 인터프리터에서 실행되도록 작성된 batch 코드
- 환경변수 설정, 프로그램 실행, 문자열 출력, 파일 처리 등이 가능
- 파일 확장자를 가지지 않거나, .sh 확장자를 가짐

```
#!/bin/bash  
clear  
ls -al
```

(셸 명령을 순차적으로 실행)

□ Shell Program

- 셸 스크립트를 이용하여 특정 기능을 구현한 코드
- 변수, 함수, 조건문, 반복문 등을 사용하여 로직 구현

```
#!/bin/bash  
echo -n "Your Age: "  
    read AGE  
if [ $AGE -gt 20 ]; then  
    echo "$AGE.. adult"  
fi
```

(흐름 제어를 활용한 명령 실행)

LINUX Shell 프로그래밍

□ Shell Programming(1)

▪ 셸 스크립트 파일 생성 절차

- ① 텍스트 편집기를 이용해 셸 스크립트 작성
- ② 확장자를 .sh로 지정하여 저장
- ③ 파일 속성의 접근권한을 실행 가능하도록(711, 755) 설정
-rw-r--r-- => -rW**X**--**X**--**X** 또는 -rW**X**r-**X**r-**X**
- ④ 셸 스크립트 실행 및 수정

```
linuxer@linuxer-PC:~$ vi name.sh
linuxer@linuxer-PC:~$ ls -l name.sh
-rw-rw-r-- 1 linuxer linuxer 70 9월 17 12:38 name.sh
linuxer@linuxer-PC:~$ chmod 711 name.sh
linuxer@linuxer-PC:~$ ls -l name.sh
-rwx--x--x 1 linuxer linuxer 70 9월 17 12:38 name.sh
linuxer@linuxer-PC:~$ ./name.sh
linuxer@linuxer-PC:~$ bash name.sh
```

LINUX Shell 프로그래밍

□ Shell Programming(2)

▪ 셸 스크립트 구조

```
#!/인터프리터  
#  
# 주석문  
#
```

```
함수명() {  
    명령들..  
}
```

명령들..

흐름제어(flow control)

함수호출

첫 줄은 스크립트를 처리할 인터프리터 정의
(shebang, 셔뱅)

#!/bin/sh	(bourn shell)
#!/bin/csh	(c shell)
#!/bin/ksh	(korn shell)
#!/bin/bash	(bash shell)
#!/usr/bin/perl	(perl interpreter)
#!/usr/bin/python	(python interpreter)

조건문(if), 반복문(for)

LINUX Shell 프로그래밍

□ Shell Programming(3)

▪ 변수 사용: \$

- 변수는 선언없이 변수명에 값을 곧바로 저장 가능
- 변수를 사용할 때는 '\$' 뒤에 변수명 지정
- 변수에 값을 배정할 때는 이퀄(=) 양쪽에 공백없이 사용
- 변수 값 확인은 **echo** 명령으로 가능

```
#!/bin/bash    (변수 활용)
a="hello, world"
echo $a

b=10
echo $b

echo "$a ... $b"
b=$a
echo "$a ... $b"
```

```
#!/bin/bash    (동일 코딩)
a="hello, world"; echo $a
b=10; echo $b; echo "$a ... $b"
b=$a; echo "$a ... $b"
```

LINUX Shell 프로그래밍

□ Shell Programming(4)

▪ 표준 입력: **read** 명령

- 키보드를 통해 값을 입력 받으려면 read 명령 사용
- 입력 받은 값은 read 명령 매개변수에 저장됨
- echo 명령은 출력 시 문자열에 개행문자를 붙여서 출력
-n 옵션을 지정하면 개행문자 없이 출력 가능

```
#!/bin/bash
```

(표준 입력 처리)

```
echo "First Value: "
```

```
read value1
```

```
echo $value1
```

```
echo -n "Second Value: "
```

```
read value2
```

```
echo -n $value2
```

LINUX Shell 프로그래밍

□ Shell Programming(5)

▪ [실습] 사용자 계정 생성용 쉘 스크립트

- 입력한 이름으로 사용자 계정을 생성하고 해당 계정이 어떻게 생성되었는지 확인
- 파일명은 myadduser.sh

```
#!/bin/bash
echo "==< 사용자가 입력한 계정 생성 예제 >=="
echo -n "생성할 계정: "
read username

useradd $username

echo "==< 생성된 계정 확인 >=="

cat /etc/passwd | grep ^$username
```

LINUX Shell 프로그래밍

□ Shell Programming(6)

- 정수 연산 : **expr** <정수값1> **연산자** <정수값2>
 - 사용 가능 연산자: +, -, *, /, %
 - 연산식은 반드시 역따옴표(`) 로 묶어줘야 함
 - 연산자, 숫자, 변수, 기호 사이에는 공백이 존재해야 함
 - 괄호를 포함해야하는 복잡한 연산식은 처리 불가

```
#!/bin/bash      (연산 활용)
a=10
b=20
c=`expr $a + $b`
echo "$a + $b = $c"

d=`expr $a - $b`
echo "$a - $b = $d"
```

```
e=`expr $a w* $b`
echo "$a * $b = $e"

f=`expr $a / $b`
echo "$a / $b = $f"

g=`expr $a % $b`
echo "$a % $b = $g"
```


LINUX Shell 프로그래밍

❑ Shell Programming(7)

▪ 실수 연산 : **bc**

- 독립 계산기로 동작시키거나 필터(|)를 활용하여 계산
- 복잡한 연산 가능: ^(power), &&(and), ||(or), !(not), 괄호 연산
- 수학 함수 라이브러리 사용 가능: **bc -l**
 - ☞ $s(x)$, $c(x)$, $a(x)$, $l(x)$, $e(x)$, ...

```
linuxer@linuxer-PC:~$ bc
2*(3.141592*2)-8
12.566360
4^4
256
sqrt(256)
16
1/6
0
a(1)
Runtime error ...
```

```
linuxer@linuxer-PC:~$ bc -l
a(1)
.78539816339744830961      (20자리)
1/6
.1666666666666666666666
scale=5; c(1)
.54030                     (5자리)
scale=10; 4*a(1)
3.1415926532
```



LINUX Shell 프로그래밍

□ Shell Programming(8)

- 단일 조건문 : **if [조건식] then <문장들> fi**
 - 조건식 = <피연산자> -<조건연산자> <피연산자>
 - 조건 연산자: -gt(>), -ge(>=), -eq(==), -lt(<), -le(<=), -ne(!=)
 - 문자열 비교: -z 문자열, -n 문자열
 - 조건식은 대괄호([]) 안에 지정하는데, 대괄호와는 공백 한 칸 띄움

```
#!/bin/bash      (단일 조건문)
echo -n "A value: "
read A

echo -n "B value: "
read B

if [ $A -gt $B ]
then
    echo "$A > $B"
fi
```

```
#!/bin/bash      (동일 코딩)
echo -n "A value: "; read A
echo -n "B value: "; read B
if [ $A -gt $B ]; then echo "$A > $B"; fi
```

LINUX Shell 프로그래밍

□ Shell Programming(9)

- 다중 조건문 : **if [조건식] then** <문장들>
elif [조건식] then <문장들>
else <문장들>
fi

```
#!/bin/bash      (다중 조건문)
echo -n "A value: "; read A
echo -n "B value; "; read B
if [ $A -gt $B ]; then
    echo "$A > $B"
elif [ $A -lt $B ]; then
    echo "$A < $B"
elif [ $A -eq $B ]; then
    echo "$A = $B"
else
    echo "error"
fi
```

```
#!/bin/bash      (동일 코딩)
echo -n "A value: "; read A
echo -n "B value; "; read B
if [ $A -gt $B ]; then echo "$A > $B"
elif [ $A -lt $B ]; then echo "$A < $B"
elif [ $A -eq $B ]; then echo "$A = $B"
else echo "error"; fi
```

※ else 문장 생략 가능



LINUX Shell 프로그래밍

□ Shell Programming(10)

- 반복문(1): **for** 변수 [**in** 변수값_목록]
do <문장들>
done

변수값_목록의 개수만큼 반복

```
#!/bin/bash
for n in 1 2 3 4 5
do
    echo $n
done
echo
for i in 1 2 3
do
    for j in 1 2 3
    do
        result=`expr $i + $j`
        echo $result
    done
done
```

```
for var in {1..5}; do
    echo var value: $var; done
```

```
for var in {1..10..2}; do
    echo var value: $var; done
```

```
for ((var=0; var<5; var++)); do
    echo var value: $var; done
```

```
for filename in `ls *.sh`; do
    ls -l $filename; done
```

```
foreach var (1 2 3 )
    echo var value: $var; end
```

LINUX Shell 프로그래밍

□ Shell Programming(11)

- 반복문(2): **while** [조건식]
 do <문장들>
 done

조건식이 참(TRUE)인 동안 반복

```
#!/bin/bash
while [ 1 ]      (while true와 동일)
do
    echo 'while test'
    sleep 1
done
```

```
#!/bin/bash
echo -n "count: "; read maxcnt
cnt=0
while [ $cnt -lt $maxcnt ]; do
    cnt=`expr $cnt + 1`
    echo 'iteration.. $cnt'
    sleep 1
done
```

```
#!/bin/bash
i=1
while [ $i -le 9 ]
do
    j=1
    while [ $j -le 9 ]
    do
        result=`expr $i \* $j`
        echo -ne "$i x $j = $result\t"
        j=`expr $j + 1`
        if [ $j -eq 6 ]; then echo; fi
    done
    echo
    i=`expr $i + 1`
done
```



LINUX Shell 프로그래밍

□ Shell Programming(12)

- 반복문(3): **until** [조건식]
do <문장들>
done

조건식이 거짓(FALSE)인 동안 반복

```
#!/bin/bash
until false
do
    echo 'until test'
    sleep 1
done
```

```
#!/bin/bash
echo -n "count: "; read maxcnt
cnt=0
until [ $cnt -ge $maxcnt ]; do
    cnt=`expr $cnt + 1`
    echo 'iteration.. $cnt'
    sleep 1
done
```

```
#!/bin/bash
i=1
until [ $i -gt 9 ]
do
    j=1
    until [ $j -gt 9 ]
    do
        result=`expr $i \* $j`
        echo -ne "$i x $j = $result\t"
        j=`expr $j + 1`
        if [ $j -eq 6 ]; then echo; fi
    done
    echo
    i=`expr $i + 1`
done
```

LINUX Shell 프로그래밍

□ Shell Programming(13)

- 패턴 검색(1): **case** 변수 **in**
 패턴1) <문장들>;
 패턴2) <문장들>;
 ***)** <문장들>;
 esac

```
#!/bin/bash
echo -n "Exit the program? "
read answer
case "$answer" in
    y) echo "exit..";;
    n) echo "no..";;
    *) echo "error..";;
esac
```

```
#!/bin/bash
case $1 in
    start) echo "service start..";;
    stop) echo "service stop..";;
    restart) echo "service restart..";;
    *) echo "invalid argument..";;
esac
```

LINUX Shell 프로그래밍

□ Shell Programming(14)

- 패턴 검색(2): **select** 변수 **in** 명령_목록
do <문장들>
done

메뉴 기반 쉘 스크립트 만들때 사용

```
#!/bin/bash
echo -n "Select Command: "
select cmd in ls ps df quit
do
case $cmd in
    ls) ls -lF;;
    ps) ps -aef | tail -3;;
    df) df -h | head -5;;
    quit) break;;
    *) echo "invalid..";;
esac
done
```

```
linuxer@linuxer-PC:~$ ./select.sh
Select Command:
1) ls
2) ps
3) df
4) quit
#?
```


LINUX Shell 프로그래밍

❑ Shell Programming(15)

- bash 디버깅: **-x 옵션**
- in line 디버깅 시작: **set -x**
- in line 디버깅 종료: **set +x**

```
linuxer@linuxer-PC:~$ bash -x ./isfile.sh
```

```
#!/bin/bash -x
echo -n "Filename: "
read filename

if [ -f $filename ]; then
echo "$filename file exists."
else
echo "$filename does not exists."
fi
```

```
#!/bin/bash
echo -n "Filename: "
read filename

if [ -f $filename ]; then
set -x
echo "$filename file exists."
set +x
else
echo "$filename does not exists."
fi
```

LINUX Shell 프로그래밍

□ [참고]

▪ Windows Batch 파일에서 isfile 구현

```
C:\W> isfile.bat
```

```
@echo off
setlocal

set /p filename="Filename: "

if exist %filename% (
    echo %filename% file exist.
) else (
    echo %filename% does not exist.
)
```



LINUX Shell 프로그래밍

□ Shell Programming(16)

- 함수 작성: [function] 함수명()
- 함수 사용: 함수명 [매개변수들]

```
#!/bin/bash
function add() {
    a=100; b=200;
    echo `expr $a + $b`
}

add
```

```
#!/bin/bash
function add() {
    a=$1; b=$2;
    echo `expr $a + $b`
}

result=`add 100 200`; echo $result
```

```
#!/bin/bash
function add() {
    a=$1; b=$2;
    echo `expr $a + $b`
}

add 100 200
```

```
#!/bin/bash
function add() {
    a=$1; b=$2;
    return `expr $a + $b`
}

add 200 55; echo $?
```

LINUX Shell 프로그래밍

- ❑ [참고] MySQL/MariaDB 데이터 조회용 셸 스크립트
 - MySQL/MariaDB는 명령행 쿼리 가능: -e 옵션
 - 데이터베이스 로그인 정보가 노출되는 문제가 있음
 - SELECT 쿼리를 명령행으로 전달 후 반환값 획득하여 결과 필터링

```
#!/bin/bash
```

```
db=mydb
```

```
user=root
```

```
pass=admin
```

```
totalUserCount="SELECT count(*) FROM tbl_user;"
```

```
echo "-----"
```

```
ret=$(mysql -u$user -p$pass $db -e "$totalUserCount" | sed 1d)
```

```
echo ">>total user count : $ret"
```

```
echo "-----"
```

LINUX Shell 프로그래밍

❑ [참고] MySQL/MariaDB 테이블 추출용 셸 스크립트

- 테이블 목록 확인 후 각 테이블에 대해 반복 추출
- 레코드는 탭(0x09)으로 필드가 분리되어 있는데 이를 콤마(,)로 변경하면 .csv 파일 생성 가능

```
#!/bin/bash
```

```
db=mydb
```

```
user=root
```

```
pass=admin
```

```
for table in $(mysql -u$user -p$pass $db -Be "SHOW tables" | sed 1d);  
do
```

```
    echo "exporting $table.."
```

```
    mysql -u$user -p$pass $db -e "SELECT * FROM $table" | sed
```

```
's/\t/,/g;' > $table.csv
```

```
done
```

```
SELECT * FROM tbl_user  
INTO OUTFILE '/tmp/tbl_user.csv'  
FIELDS TERMINATED BY ','  
ENCLOSED BY ''  
LINES TERMINATED BY '\n'
```