UNIX 및 실습

13장 보충 – bash(1)

Bash

- bash(Bourne Again Shell)
  - 다양한 내장 명령과 히스토리, 별명, 파일, 명령완성, 명령줄 편집 등 지원
  - 원래 있던 Bourne Shell에 GNU 프로젝트를 통해 추가된 다양한 기능들이 많음
  - 버전 확인

```
[kgu@lily ~]$ bash --version
GNU bash, version 4.2.37(1)-release (x86_64-redhat-linux-gnu)
Copyright (C) 2011 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later
<http://gnu.org/licenses/gpl.html>

This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

환경

- 초기화 파일
  - /etc/profile
  - ~/.bash_profile
    - 없으면 ~/.bash_login
    - 그마저도 없으면 ~/.profile
    - 3개 중 하나만 처리
  - .bashrc
    - Bash 쉘에만 적용되는 특별한 설정 사항들
  - /etc/bashrc
    - 시스템 전체에서 사용되는 함수와 별명들

변수 확장변경자(modifier)

서식 지정자	값
${variable:-word}	변수에 null 이 아닌 값이 지정되어 있으면 변수값 사용, 그렇지 않으면 word로 잠시 치환
${variable:=word}	변수에 null 이 아닌 값이 지정되어 있으면 변수값 사용, 그렇지 않으면 word로 설정 (영구 치환)
${variable:+word}	변수에 null 이 아닌 값이 지정되어 있으면 변수값을 word로 잠시 치환, 그렇지 않으면 아무 것도 치환하지 않는다.
${variable:?word}	변수에 null 이 아닌 값이 지정되어 있으면 변수값 사용, 그렇지 않으면 word를 출력하고 쉘 종료. word를 생략하면 parameter null or not set 메시지 출력
${variable:offset}	offset에서 시작하여 변수 variable의 부분 문자열 추출.
${variable:offset:length}	offset에서 시작하여 length 만큼 변수 variable의 부분 문자열 추출.

부분문자열에 대한 변수 전개

표현식	기능
${variable%pattern}	Variable 값의 후반부에서 패턴과 일치하는 가장 작은 부분을 찾아 제거
${variable%%pattern}	Variable 값의 후반부에서 패턴과 일치하는 가장 큰 부분을 찾아 제거
${variable#pattern}	Variable 값의 전반부에서 패턴과 일치하는 가장 작은 부분을 찾아 제거
${#variable}	변수의 문자수로 치환. *나 @를 사용하면 위치매개변수의 개수를 길이로 사용

위치 매개변수

위치 매개변수	의미
$0	현재 쉘 스크립트 이름
$1 - $9	1번부터 9번까지 위치 매개변수
${10}	10번 위치 매개변수
$#	위치매개변수 총개수
$*	모든 위치매개변수
$@	큰 따옴표를 사용하였을 때를 제외하고는 $*와 동일
"$*"	"$1 $2 $3"로 평가
"$@"	"$1" "$2" "$3"으로 평가

기타 특수변수

특수변수	의미
$	쉘의 PID
-	쉘의 현재 설정된 옵션
?	최근에 실행시킨 명령의 종료상태
!	백그라운드에서 실행시킨 최근 프로그램의 PID

인용부호의 사용 (1)

메타문자	의미
;	명령 구분자
&	백그라운드 실행
( )	명령 집합 : 새로운 쉘에서 실행
{ }	명령 집합 : 현재 쉘에서 실행
|	파이프

인용 부호를 필요로 하는 메타문자

메타문자	의미
<	입력 리다이렉션
>	출력 리다이렉션
Newline	명령 입력 종료
Space/TAB	단어 구분자
$	변수 치환 문자
* [ ] ?	파일명 전개를 위한 쉘 메타문자

인용부호 사용 (2)

- 역슬래시 (₩)
  - 어떤 문자가 쉘에 의해 해석되지 않도록 보호
- 작은 따옴표 (‘ ’)
  - 쌍으로 사용
  - 모든 메타문자의 해석을 막는다
- 큰 따옴표 (“ ”)
  - 쌍으로 사용
    - 변수나 명령 치환은 허용하고 나머지 특수메타문자들에 대해서는 쉘이 해석하지 않도록 보호

Shell Built-in Commands (1)

명령	기능
:	do-nothing; 0 반환
.file	명령을 File에서 읽어 수행
break [n]	반복문 참조
.	현재 프로세스에서 프로그램 수행 (source와 동일)
alias	현재 존재하는 명령들에 대한 별명들을 보여주거나 설정
bg	작업을 백그라운드화
bind	현재 설정된 키와 기능 바인딩을 보여주거나, readline이나 매크로에서 사용할 수 있도록 키들을 바인딩
break	가장 안에 있는 루프에서 탈출
built-in [*sh-built-in* [*args*]]	쉘 빌트인을 수행(args들을 넘겨주고 0 반환) built-in과 sh-built-in이 동일한 이름일 경우 유용
cd [arg]	디렉토리 변경
command command [*arg*]	함수와 명령이 동일한 이름을 가지더라도 명령을 수행
continue [*n*]	반복문 참조
declare [*var*]	모든 변수들을 보여주거나 변수 선언

Shell Built-in Commands (2)

명령	기능
dirs	pushd로 인해 현재 기억된 디렉토리 리스트를 보여줌
disown	작업 테이블에서 실행 중인 작업 제거
echo [*args*]	*args*를 보이고 줄을 바꿈
enable	쉘 빌트인 명령을 가용 또는 불용하게 함
eval [*args*]	args를 쉘의 입력으로 읽어 실행
exec *command*	현재 쉘에서 명령 수행
exit [*n*]	종료 상태를 *n*으로 반환하고 종료
export [*var*]	자식 쉘에게 var 상속
fc	히스트리 목록을 편집하기 위한 명령 (**f**ix **c**ommand)
fg	백그라운드 작업을 포그라운드로 가져옴
getopts	명령줄에서 지정한 옵션을 추출하기 위해 스크립트에서 사용
hash	명령의 빠른 수행을 위해 내부 해시 테이블 제어
help [*command*]	내장 명령에 대한 도움말 제공
history	히스토리 목록을 행번호와 같이 출력

Shell Built-in Commands (3)

명령	기능
jobs	백그라운드 작업 목록을 보여줌
kill [*-signal process*]	지정된 PID 번호나 작업번호에 시그널을 보냄
let	산술식을 평가하거나 산술 계산 결과를 변수에 저장하기 위해 사용
local	변수의 통용범위를 함수 내로 제한하기 위해 사용
logout	로그린 쉘 종료
popd	디렉토리 스택의 항목을 제거
pushd	디렉토리 스택에 항목 추가
pwd	현재 작업 디렉토리를 보여줌
read [*var*]	표준 입력에서 읽은 내용을 변수 *var*에 저장
readonly [*var*]	변수 *var*를 읽기 전용으로 만듬 (*var*은 재설정이 안됨)
return [*n*]	종료 상태를 *n*으로 반환하고 함수로부터 복귀
set	옵션과 위치 매개변수 설정
shift [*n*]	위치 매개변수를 지정한 *n*회만큼 왼쪽으로 이동
stop *pid*	지정된 PID 프로세스를 중지(halt)

Shell Built-in Commands (4)

명령	기능
suspend	현재 쉘의 수행을 일시 중지
test	파일 유형이나 조건식을 평가
times	현재 쉘에서 수행된 프로세스들의 수행시간에 대한 정보를 출력
trap [*arg*] [*n*]	쉘은 시그널 *n*에 대해 *arg*를 수행
type [*command*]	명령의 유형을 출력
typeset	변수의 설정이나 속성을 지정, declare와 동일
ulimit	프로세스가 사용할 수 있는 자원의 최대 한계를 설정
umask [*octal_digits*]	파일 생성시에 사용할 권한을 설정
unalias	별명의 설정을 해제
unset [*name*]	변수나 함수의 설정을 해제
wait [pid#n]	백스라운드에서 수행되는 지정한 PID의 프로세스가 종료할 때까지 스크립트를 일시 중지

쉘 스크립트 작성 절차

- 첫 번째 행에는 스크립트를 수행할 때 사용할 쉘 이름
  - #!/bin/bash
    - #!는 매직 넘버 (항상 첫 줄에 나와야 함)
- 주석
  - # 기호가 앞서 나오는 줄
- 실행문과 배쉬 쉘 구조
  - 유닉스 명령, 본쉘 명령, 프로그램 구조, 주석 등으로 구성
- 실행 권한 부여
  - 반드시 실행 권한을 부여해야만 실행 가능

예제 쉘 스크립트

```
[kgu@lily ch13]$ cat greeting.bash
#!/bin/bash
# This is the first Bash shell program of the day.
# Scriptname: greetings
# Written by:  Barbara Bashful
echo "Hello $LOGNAME, it's nice talking to you."
echo "Your present working directory is `pwd`."
echo "You are working on a machine called `uname -n`."
echo "Here is a list of your files."
ls            # List files in the present working directory
echo  "Bye for now $LOGNAME. The time is `date +%T`!"
```

```
[kgu@lily ch13]$ chmod +x greeting.bash
[kgu@lily ch13]$ ./greeting.bash
Hello kgu, it's nice talking to you.
Your present working directory is /home/kgu/2013U1/ch13.
You are working on a machine called lily.mmu.ac.kr.
Here is a list of your files.
greeting.bash
Bye for now kgu. The time is 17:37:30!
```

사용자 입력 읽기 (1)

- read 명령

형식	의미
read answer	표준 입력에서 한 줄 읽어서 변수 answer에 저장
read first last	표준 입력에서 한 줄 읽어서 첫 단어는 first에, 나머지는 last에 저장
read	입력을 REPLY에 저장
read –a arrayname	입력을 받은 목록을 arrayname 배열에 저장
read -e	입력 줄에서 명령줄 편집 사용 입력 줄을 vi 키를 이용하여 편집
read –p prompt	문자열 prompt를 출력하고 입력을 기다린다. 입력은 REPLY 변수에 저장
read –r line	입력에 역슬래시를 쓸 수 있게 허용

사용자 입력 읽기 (2)

```
[kgu@lily ch13]$ cat nosy.bash
#!/bin/bash
# Scriptname: nosy
echo -e "Are you happy? \c"
read answer
echo "$answer is the right response."
echo -e "What is your full name? \c"
read first middle last
echo "Hello  $first"
echo -n "Where do you work? "
read
echo I guess $REPLY keeps you busy!
read -p "Enter your job title: "
echo "I thought you might be an $REPLY."

echo -n "Who are your best friends? "
read -a friends
echo "Say hi to ${friends[2]}."
```

```
[kgu@lily ch13]$ chmod +x nosy.bash
[kgu@lily ch13]$ ./nosy.bash
Are you happy? y
y is the right response.
What is your full name? kgu
Hello  kgu
Where do you work? MMU
I guess MMU keeps you busy!
Enter your job title: Prof.
I thought you might be an Prof..
Who are your best friends? kki kjw kjh
Say hi to kjh.
```

사용자 입력 읽기 (3)

```
[root@lily ch13]# cat printer_check.bash
#!/bin/bash
# Scriptname: printer_check
# Script to clear a hung-up printer
if [ $LOGNAME != root ]
then
            echo "Must have root privileges to run this program"
            exit 1
fi
cat << EOF
  Warning: All jobs in the printer queue will be removed.
  Please turn off the printer now. Press return when you
  are ready to continue. Otherwise press Control C.
EOF
read JUNK      # Wait until the user turns off the printer
echo
/etc/rc.d/init.d/lpd stop       # Stop the printer
echo -e "\nPlease turn the printer on now."
echo "Press Enter to continue"
read JUNK                # Stall until the user turns the printer
                         # back on
echo                     # A blank line is printed
/etc/rc.d/init.d/lpd start     # Start the printer
```

산술 연산 (1)

- declare 명령
  - declare –i로 정수 타입 변수 선언

```
[kgu@lily ch13]$ declare -i number
[kgu@lily ch13]$ number=hello
[kgu@lily ch13]$ echo $number
0
```

```
[kgu@lily ch13]$ number=5 + 5
bash: +: command not found...

[kgu@lily ch13]$ number=5+5
[kgu@lily ch13]$ echo $number
10
[kgu@lily ch13]$ number=6.5
-bash: 6.5: syntax error: invalid arithmetic operator (error token is ".5")
```

산술 연산 (2)

정수 타입 변수 출력

```
[kgu@lily ch13]$ declare -i
declare -ir BASHPID
declare -ir EUID="1000"
declare -i HISTCMD
declare -i LINENO
declare -i MAILCHECK="60"
declare -i OPTIND="1"
declare -ir PPID="6929"
declare -i RANDOM
declare -ir UID="1000"
declare -i num="0"
declare -i number="10"
```

다른 수 체계 사용

```
[kgu@lily ch13]$ declare -i x=017
[kgu@lily ch13]$ echo $x
15
[kgu@lily ch13]$ x=2#101
[kgu@lily ch13]$ echo $x
5
[kgu@lily ch13]$ x=8#17
[kgu@lily ch13]$ echo $x
15
```

2진수 101

8진수 17

산술 연산 (3)

- let 명령

```
[kgu@lily ch13]$ i=5
[kgu@lily ch13]$ let i=i+1
[kgu@lily ch13]$ echo $i
6
[kgu@lily ch13]$ let "i = i + 2"
[kgu@lily ch13]$ echo $i
8
[kgu@lily ch13]$ let "i+=1"
[kgu@lily ch13]$ echo $i
9
[kgu@lily ch13]$ i=3
[kgu@lily ch13]$ (i+=4)
[kgu@lily ch13]$ echo $i
3
[kgu@lily ch13]$ ((i+=4))
[kgu@lily ch13]$ echo $i
7
```

- 부동소숫점 연산
  - Bash는 정수타입 연산만 지원
  - bc, awk, nawk 등을 이용하여 복잡한 계산 수행

```
[kgu@lily ch13]$ n='echo "scale=3; 13/2" |bc'
[kgu@lily ch13]$ n=`echo "scale=3; 13/2" |bc`
[kgu@lily ch13]$ echo $n
6.500
[kgu@lily ch13]$ product=`gawk -v x=2.45 -v y=3.123
'BEGIN { printf "%.2f\n", x * y }'`
[kgu@lily ch13]$ echo $product
7.65
```

위치 매개변수 이용 (1)

```
[kgu@lily ch13]$ cat greeting2.bash
#!/bin/bash
# Scriptname: greetings2
echo "This script is called $0."
echo  "$0  $1 and $2"
echo "The number of positional parameters is $#"
[kgu@lily ch13]$ chmod +x greeting2.bash
[kgu@lily ch13]$ ./greeting2.bash
This script is called ./greeting2.bash.
./greeting2.bash   and
The number of positional parameters is 0
[kgu@lily ch13]$ ./greeting2.bash Tommy
This script is called ./greeting2.bash.
./greeting2.bash  Tommy and
The number of positional parameters is 1
[kgu@lily ch13]$ ./greeting2.bash Tommy Billy
This script is called ./greeting2.bash.
./greeting2.bash  Tommy and Billy
The number of positional parameters is 2
```

set으로 설정 가능

```
[kgu@lily ch13]$ cat args.bash
#!/bin/bash
# Scriptname: args
# Script to test command line arguments
echo The name of this script is $0.
echo The arguments are $*.
echo The first argument is $1.
echo The second argument is $2.
echo The number of arguments is $#.
oldargs=$*
set Jake Nicky Scott    # Reset the positional parameters
echo All the positional parameters are $*.
echo The number of postional parameters is $#.
echo "Good-bye for now, $1."
set $(date)             # Reset the positional parameters
echo The date is $2 $3, $6.
echo "The value of \$oldargs is $oldargs."
set $oldargs
echo $1 $2 $3
[kgu@lily ch13]$ chmod +x ./args.bash
[kgu@lily ch13]$ ./args.bash a b c d
The name of this script is ./args.bash.
The arguments are a b c d.
The first argument is a.
The second argument is b.
The number of arguments is 4.
All the positional parameters are Jake Nicky Scott.
The number of postional parameters is 3.
Good-bye for now, Jake.
The date is 05. 27., KST.
The value of $oldargs is a b c d.
a b c
```

위치 매개변수 이용 (2)

```
[kgu@lily ch13]$ cat checker.bash
#!/bin/bash
# Scriptname: checker
# Script to demonstrate the use of special
    variable
# modifiers and arguments
name=${1:?"requires an argument" }
echo Hello $name
[kgu@lily ch13]$ chmod +x checker.bash
[kgu@lily ch13]$ ./checker.bash
./checker.bash: line 5: 1: requires an argument
[kgu@lily ch13]$ ./checker.bash kgu
Hello kgu
```

```
[kgu@lily ch13]$ set 'apple pie' pears peaches
[kgu@lily ch13]$ for i in $*
> do
> echo $i
> done
apple
pie
pears
peaches
[kgu@lily ch13]$ set 'apple pie' pears peaches
[kgu@lily ch13]$ for i in "$*"
> do
> echo $i
> done
apple pie pears peaches
[kgu@lily ch13]$ set 'apple pie' pears peaches
[kgu@lily ch13]$ for i in $@
> do
> echo $i
> done
apple
pie
pears
peaches
[kgu@lily ch13]$ set 'apple pie' pears peaches
[kgu@lily ch13]$ for i in "$@"
> do
> echo $i
> done
apple pie
pears
peaches
```

조건의 표현과 흐름 제어 (1)

```
[kgu@lily ch13]$ name=Tom
[kgu@lily ch13]$ grep "$name" /etc/passwd
[kgu@lily ch13]$ echo $?
1
[kgu@lily ch13]$ name=kgu
[kgu@lily ch13]$ grep "$name" /etc/passwd
kgu:x:1000:1000:kgu:/home/kgu:/bin/bash
[kgu@lily ch13]$ echo $?
0
```

- 표현식 평가
  - 단일 대괄호와 test
    - 쉘은 메타문자를 전개하지 않음
    - 단어 단위로 나누어 처리하므로 스페이스를 포함한 문자열은 반드시 따옴표로 묶어야 함
  - 이중 대괄호와 test
    - 패턴검색, 메타문자에 대한 해석 가능
    - 스페이스를 포함한 문자열은 반드시 따옴표로 묶어야 함
    - 정확히 일치하는지를 검사할 경우 따옴표 이용
    - 논리연사자 사용 가능

조건의 표현과 흐름 제어 (2)

- test 명령

검사연산자	참인 경우
[string1 = string2] [string1 == string2]	string1과 string2는 같다 (양쪽에 스페이스를 이용해야 함) Bash 2.x 이후에는 == 이용 가능
[string1 != string2]	string1과 string2는 다르다
[string]	string은 null이 아니다
[-z string]	string 길이가 0이다
[-n string]	string 길이가 0이 아니다

검사연산자	참인 경우
[string1 –a string2]	and 연산자 (string1, string2 모두 참)
[string1 -o string2]	or 연산자 (string1과 string2 둘 중 하나 이상 참)
[!string]	not 연산자

검사연산자	참인 경우
[[pattern1 && pattern2]]	pattern1, pattern2 모두 참 (정확한 일치 여부 확인시 "pattern2")
[[pattern1 || pattern2]]	pattern1, pattern2 모둘 중 하나 이상 참
[!pattern]	pattern과 같지 않다

조건의 표현과 흐름 제어 (3)

- test 명령 (계속)

검사연산자	참인 경우
[int1 –eq int2]	int1과 int2는 같다
[int1 –ne int2]	int1과 int2는 같지 않다
[int1 –gt int2]	int1은 int2보다 크다
[int1 –ge int2]	int1은 int2보다 크거나 같다
[int1 –lt int2]	int1은 int2보다 작다
[int1 –le int2]	int1은 int2보다 작거나 같다

검사연산자	참인 경우
[file1 –nt file2]	file1이 file2보다 새로운 파일 (변경일자 기준)
[file1 –ot file2]	file1이 file2보다 오래 된 파일
[file1 –et file2]	file1이 file2과 동일한 장치이거나 같은 inode를 가짐

조건의 표현과 흐름 제어 (4)

- let 명령
  - C 언어와 같이 풍부한 연산자 사용 가능
  - Bash 2.x 이후 ( ), (( ))로 치환하여 사용 가능
  - 주의! 종료가 0이면 성공, 1이면 실패

```
[kgu@lily ch13]$ x=2
[kgu@lily ch13]$ y=3
[kgu@lily ch13]$ (( x > 2 ))
[kgu@lily ch13]$ echo $?
1
[kgu@lily ch13]$ (( x < 2 ))
[kgu@lily ch13]$ echo $?
1
[kgu@lily ch13]$ (( x <= 2 ))
[kgu@lily ch13]$ echo $?
0
```

조건의 표현과 흐름 제어 (5)

if 명령

```
if 명령
then
     명령
fi
-----------------------------------------
문자열에 대해 test 사용 - 신 양식 [[ ]]
if [[ 문자열 조건식 ]] then
     명령
fi
숫자에 대해 let 사용 - 신양식 (( ))
if (( 수식 )) then
     명령
fi
```

```
[kgu@lily ch13]$ echo  "Are you o.k. (y/n) ?"
Are you o.k. (y/n) ?
[kgu@lily ch13]$ read answer
y
[kgu@lily ch13]$ if [ "$answer" = Y -o "$answer" = y ]
> then
> echo "Glad to hear it."
> fi
Glad to hear it.
[kgu@lily ch13]$ if [[ $answer == [Yy]* || $answer == Maybe ]]
> then
> echo  "Glad to hear it."
> fi
Glad to hear it.
```

```
if 명령
then
     명령(들)
else
     명령(들)
fi
```

```
if 명령
then
     명령(들)
elif 명령
then
     명령(들)
elif 명령
then
     명령(들)
else
     명령(들)
fi
```

조건의 표현과 흐름 제어 (6)

- 파일 검사연산자

검사연산자	참인 경우	검사연산자	참인 경우
-b filename	블록 파일	-p filename	named pipe
-c filename	문자 파일	-O filename	파일이 존재하고 유효 사용자 ID 소유임
-d filename	디렉토리가 존재	-S filename	소켓
-e filename	파일이 존재	-s filename	파일 크기가 0이 아니다
-f filename	파일이 존재하고 디렉토리가 아님	-t fd	파일식별자가 터미널에 열려있음
-G filename	파일이 존재하고 유효 그룹 ID 소유	-u filename	setuid가 설정됨
-g filename	setgid가 설정됨	-w filename	쓰기 가능
-k filename	Sticky bit이 설정됨	-x filename	실행 가능
-L filename	심볼릭 링크		

조건의 표현과 흐름 제어 (7)

```
[kgu@lily ch13]$ cat perm_check.bash
#!/bin/bash
# Using the old style test command
# filename: perm_check
file=./testing
if [ -d $file ]
then
     echo "$file is a directory"
elif [ -f $file ]
then
     if [ -r $file -a -w $file -a -x $file ]
     then                 # nested if command
               echo "You have read,write,and execute \
               permission on $file."
     fi
else
     echo "$file is neither a file nor a directory.
     "
fi
[kgu@lily ch13]$ chmod +x perm_check.bash
[kgu@lily ch13]$ ./perm_check.bash
./testing is neither a file nor a directory.
```

```
[kgu@lily ch13]$ cat perm_check2.bash
#!/bin/bash
# Using the new compound operator for test (( ))
# filename: perm_check2

file=./testing

if [[ -d $file ]]
then
     echo "$file is a directory"
elif [[ -f $file ]]
then
     if [[ -r $file && -w $file && -x $file ]]
     then      # nested if command
               echo "You have read,write,and execute \
               permission on $file."
     fi
else
     echo "$file is neither a file nor a directory.
     "
fi
[kgu@lily ch13]$ chmod +x perm_check2.bash
[kgu@lily ch13]$ ./perm_check2.bash
./testing is neither a file nor a directory.
```

조건의 표현과 흐름 제어 (8)

- Null 명령
  - 내장 명령으로서 :으로 나타냄
  - 실제로 아무런 작업을 하지 않으며, 단지 종료 상태를 0으로 돌려줌
  - 주로 if 다음에 아무런 작업을 하지 않을 때 이용

```
[kgu@lily ch13]$ cat name_grep.bash
#!/bin/bash
# filename: name_grep

name=Tom
if grep "$name" databasefile >& /dev/null
then
     :
else
     echo "$1 not found in databasefile"
     exit 1
fi
[kgu@lily ch13]$ chmod +x name_grep.bash
[kgu@lily ch13]$ ./name_grep.bash
 not found in databasefile
```

```
[kgu@lily ch13]$ cat wholenum.bash
#!/bin/bash
# Scriptname: wholenum
# Purpose:The expr command tests that the
     user enters an
# integer
#
echo "Enter an integer."
read number
if expr "$number" + 0 >& /dev/null
then
     :
else
     echo "You did not enter an integer
     value."
     exit 1
fi
[kgu@lily ch13]$ chmod +x wholenum.bash
[kgu@lily ch13]$ ./wholenum.bash
Enter an integer.
as
You did not enter an integer value.
[kgu@lily ch13]$ ./wholenum.bash
Enter an integer.
10
```

조건의 표현과 흐름 제어 (9)

- case 명령
  - 다중 분기명령으로 if/elif 대신 사용 가능
  - default 행동은 *)
  - 형식

```
case 변수 in
value1)
     명령(들)
     ;;
value2)
     명령(들)
     ;;
*)
명령(들)
     ;;
esac
```

```
[kgu@lily ch13]$ cat xcolors.bash
#!/bin/bash
# Scriptname: xcolors
echo -n "Choose a foreground color for your xterm
     window: "
read color
case "$color" in
[Bb]l??)
     xterm -fg blue -fn terminal &
     ;;
[Gg]ree*)
     xterm -fg darkgreen -fn terminal &
     ;;
red | orange)              # The vertical bar
     means "or"
     xterm -fg "$color"  -fn terminal &
     ;;
*)
     xterm -fn terminal
     ;;
esac
echo  "Out of case command"
[kgu@lily ch13]$ chmod +x xcolors.bash
[kgu@lily ch13]$ ./xcolors.bash
Choose a foreground color for your xterm window:
     white
xterm: Xt error: Can't open display:
xterm: DISPLAY is not set
Out of case command
```

조건의 표현과 흐름 제어 (10)

- here 문서와 case 명령
  - 주로 함께 사용
  - here 문서로 메뉴 구성
  - case에서 사용자 선택 검사

```
[kgu@lily ch13]$ cat here_test.bash
#!/bin/bash
# Scriptname: here_test
echo "Select a terminal type: "
cat << ENDIT
     1) unix
     2) xterm
     3) sun
ENDIT
read choice
case "$choice" in
1)
     TERM=unix
     export TERM
     ;;
2) TERM=xterm
     export TERM
     ;;
3) TERM=sun
     export TERM
     ;;
esac
echo "TERM is $TERM."
[kgu@lily ch13]$ chmod +x here_test.bash
[kgu@lily ch13]$ ./here_test.bash
Select a terminal type:
     1) unix
     2) xterm
     3) sun
1
TERM is unix.
```

[실습과제]

- 실습 각 단계 화면 캡처하여 pdf 파일로 정리하여 과제 제출 (cms.mmu.ac.kr/bear)
- 제출기한 : 6월 6일 자정