

LINUX

Autumn 2025



Daejeon Univ.
Seongbok Baik
sbbaik@dju.ac.kr

Text Book

교재명	우분투 리눅스 시스템 & 서버
저자	창병모
출판사	생능출판사
발행년	2024.07.12
ISBN	979-11-92932-72-9

우분투 리눅스 시스템 & 서버

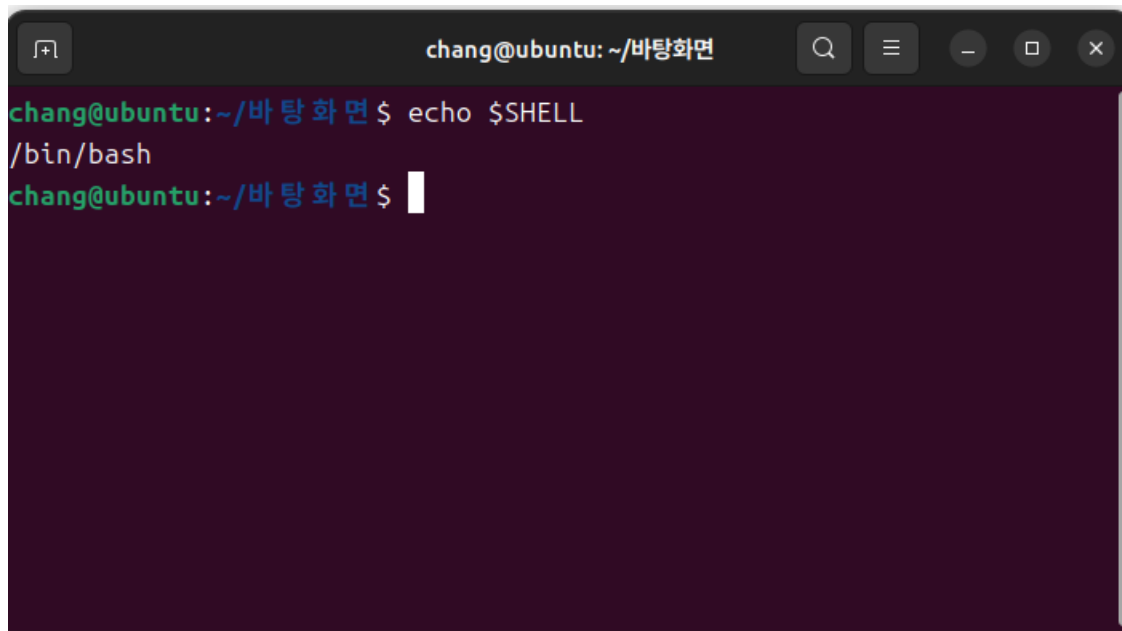


제8장 Bash 셸 스크립트

8.1 Bash 셸

Bash(Borune-again shell)

- 리눅스, 맥 OS X 등의 운영 체제의 기본 쉘
- Bash 문법은 본 쉘의 문법을 대부분 수용하면서 확장
- 쉘의 핵심 기능, 히스토리, 별명, 작업 제어

A screenshot of a terminal window titled "chang@ubuntu: ~/바탕화면". The window has a dark background and a light-colored text. The prompt "chang@ubuntu:~/바탕 화면\$" is shown in green. The user has entered the command "echo \$SHELL", and the output "/bin/bash" is displayed in white. The prompt "chang@ubuntu:~/바탕 화면\$" is shown again with a cursor at the end.

```
chang@ubuntu:~/바탕 화면$ echo $SHELL
/bin/bash
chang@ubuntu:~/바탕 화면$
```

시작 파일(start-up file)

- `/etc/profile`
전체 사용자에게 적용되는 환경 설정, 시작 프로그램 지정
- `/etc/bashrc`
전체 사용자에게 적용되는 별명과 함수들을 정의
- `~/.bash_profile`
각 사용자를 위한 환경을 설정, 시작 프로그램 지정
- `~/.bashrc`
각 사용자를 위한 별명과 함수들을 정의

셸의 종류	시작 파일 종류	시작파일 이름	실행 시기
본 셸	시스템 시작 파일	<code>/etc/profile</code>	로그인
	사용자 시작 파일	<code>~/.profile</code>	로그인
Bash 셸	시스템 시작 파일	<code>/etc/profile</code>	로그인
	시스템 시작 파일	<code>/etc/bashrc</code>	로그인, 서버셸
	사용자 시작 파일	<code>~/.bash_profile</code>	로그인
	사용자 시작 파일	<code>~/.bashrc</code>	로그인, 서버셸

Bash 시작 과정

```
/etc/profile
|
/etc/bash.bashrc
|
~/.bash_profile
|
~/.bashrc
|
로그인 셸 프롬프트
```

시작 파일 예: /etc/profile



```
$ more /etc/profile
```

```
# /etc/profile: system-wide .profile file for the Bourne shell (sh(1))
```

```
# and Bourne compatible shells (bash(1), ksh(1), ash(1), ...).
```

```
if [ "${PS1-}" ]; then
```

```
  if [ "${BASH-}" ] && [ "$BASH" != "/bin/sh" ]; then
```

```
    # The file bash.bashrc already sets the default PS1.
```

```
    # PS1='Wh:WwW$ '
```

```
    if [ -f /etc/bash.bashrc ]; then
```

```
      . /etc/bash.bashrc
```

```
    fi
```

```
  else
```

```
    if [ "$(id -u)" -eq 0 ]; then
```

```
      PS1='# '
```

```
    else
```

```
      PS1='$ '
```

```
    fi
```

```
  fi
```

```
fi
```


시작 파일 예: /etc/bash.bashrc



```
$ more /etc/bash.bashrc
# System-wide .bashrc file for interactive bash(1) shells.
# To enable the settings / commands in this file for login shells as well,
# this file has to be sourced in /etc/profile.

# If not running interactively, don't do anything
[ -z "$PS1" ] && return

# set variable identifying the chroot you work in (used in the prompt below)
if [ -z "${debian_chroot:-}" ] && [ -r /etc/debian_chroot ]; then
    debian_chroot=$(cat /etc/debian_chroot)
fi

# set a fancy prompt (non-color, overwrite the one in /etc/profile)
# but only if not SUDOing and have SUDO_PS1 set; then assume smart user.
if ! [ -n "${SUDO_USER}" -a -n "${SUDO_PS1}" ]; then
    PS1='${debian_chroot:+($debian_chroot)}wu@wh:www$ '
fi
...
```

시작 파일 예: ~/.bash_profile



```
$ more ~/.profile
```

```
# ~/.profile: executed by the command interpreter for login shells.
```

```
# This file is not read by bash(1), if ~/.bash_profile or ~/.bash_login exists.
```

```
# if running bash
```

```
if [ -n "$BASH_VERSION" ]; then
```

```
    # include .bashrc if it exists
```

```
if [ -f "$HOME/.bashrc" ]; then
```

```
    . "$HOME/.bashrc"
```

```
fi
```

```
fi
```

```
# set PATH so it includes user's private bin if it exists
```

```
if [ -d "$HOME/bin" ] ; then
```

```
    PATH="$HOME/bin:$PATH"
```

```
fi
```

```
PATH=".:$PATH"
```

```
# set PATH so it includes user's private bin if it exists
```

```
if [ -d "$HOME/.local/bin" ] ; then
```

```
    PATH="$HOME/.local/bin:$PATH"
```

```
fi
```

시작 파일 예: ~/.bashrc



```
$ more ~/.bashrc
# ~/.bashrc: executed by bash(1) for non-login shells.
# If not running interactively, don't do anything
case $- in
    *i*) ;;
    *) return;;
esac

# don't put duplicate lines or lines starting with space in the history.
# See bash(1) for more options
HISTCONTROL=ignoreboth

# append to the history file, don't overwrite it
shopt -s histappend

# for setting history length see HISTSIZE and HISTFILESIZE in bash(1)
HISTSIZE=1000
HISTFILESIZE=2000
...
```

시작 파일 예: ~/.bashrc

```
# set a fancy prompt (non-color, unless we know we "want" color)
case "$TERM" in
xterm-color|*-256color) color_prompt=yes;;
esac

...

# some more ls aliases
alias ll='ls -aIF'
alias la='ls -A'
alias l='ls -CF'
```

8.2 별명 및 히스토리 기능

별명

- alias 명령어
 - 문자열이 나타내는 기존 명령에 대해 새로운 이름을 별명으로 정의
- ```
$ alias 이름=문자열
$ alias dir='ls -aF'
$ dir
$ alias h=history
$ alias ll='ls -l'
```
- 현재까지 정의된 별명들을 확인
- ```
$ alias      # 별명 리스트
alias dir='ls -aF'
alias h=history
alias ll='ls -l'
```
- 이미 정의된 별명 해제
- ```
$ unalias 단어
```

# 히스토리

- 입력된 명령들을 기억하는 기능

`$ history [-rh] [번호]`

- 기억할 히스토리의 크기

`$ HISTSIZE=100`

- 로그아웃 후에도 히스토리가 저장되도록 설정

`$ HISTFILESIZE=100`

```
$ history
```

```
1 ls
```

```
2 who
```

```
3 env
```

```
4 vi test.sh
```

```
5 chmod +x test.sh
```

```
6 test.sh
```

```
7 ls
```

```
8 date
```

```
9 history
```

```
...
```

# 재실행

| 형태          | 의미                        |
|-------------|---------------------------|
| !!          | 바로 전 명령 재실행               |
| !n          | 이벤트 번호가 n인 명령 재실행         |
| ! 시작스tring  | 시작스tring으로 시작하는 최후 명령 재실행 |
| !? 서브스tring | 서브스tring을 포함하는 최후 명령 재실행  |

## ● 예

```

$!! # 바로 전 명령 재실행
$!20 # 20번 이벤트 재실행
$!gcc # gcc로 시작하는 최근 명령 재실행
$!?test.c # test.c를 포함하는 최근 명령 재실행

```



## 8.3 변수

## 단순 변수(simple variable)

- 하나의 값(문자열)만을 저장할 수 있는 변수  
\$ 변수이름=문자열  
\$ city=seoul
- 변수의 값 사용  
\$ echo \$city  
seoul
- 변수에 어느 때나 필요하면 다른 값을 대입  
\$ city=puhan
- 한 번에 여러 개의 변수를 생성  
\$ country=korea city=seoul

## 단순 변수

- 한글 문자열을 값으로 사용

```
$ country=대한민국 city=서울
```

```
$ echo $country $city
```

대한민국 서울

- 따옴표를 이용하여 여러 단어로 구성된 문자열 저장 가능

```
$ address="서울시 용산구"
```

## 리스트 변수(list variable)

- 한 변수에 여러 개의 값(문자열)을 저장할 수 있는 변수  
\$ 이름=( 문자열리스트 )

\$ cities=(서울 부산 목포)

- 리스트 변수 사용

| 리스트 사용                                                 | 의미                   |
|--------------------------------------------------------|----------------------|
| <code>\${name[i]}</code>                               | 리스트 변수 name의 i번째 원소  |
| <code>\${name[*]}</code><br><code>\${name[@]}</code>   | 리스트 변수 name의 모든 원소   |
| <code>\${#name[*]}</code><br><code>\${#name[@]}</code> | 리스트 변수 name 내의 원소 개수 |

## 리스트 변수 사용 예

- 리스트 변수 사용  
`$ echo ${cities[*]}`  
서울 부산 목포  
`$ echo ${cities[1]}`  
부산
- 리스트의 크기  
`$ echo ${#cities[*]}`      # 리스트 크기  
3  
`$ echo ${cities[3]}`
- 리스트 변수에 새로운 도시 추가  
`$ cities[3]=제주`  
`$ echo ${cities[3]}`  
제주

# 표준입력 읽기

- read 명령어
  - 표준입력에서 한 줄을 읽어서 단어들을 변수들에 순서대로 저장
  - 마지막 변수에 남은 단어들 모두 저장

**\$ read 변수1 ... 변수n**

```
$ read x y
Merry Christmas !
$ echo $x
Merry
$ echo $y
Christmas !
```

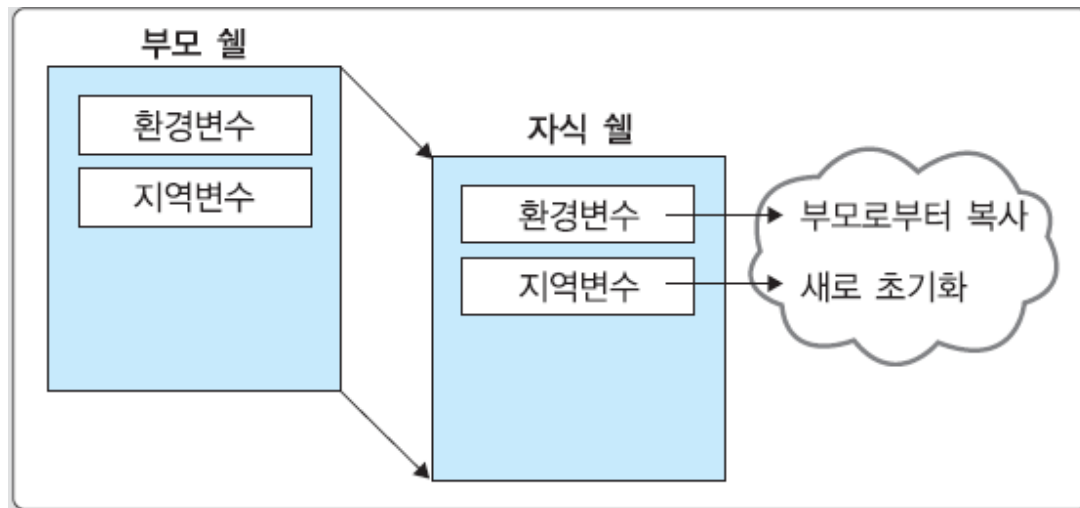
- 변수를 하나만 사용

```
$ read x
Merry Christmas !
$ echo $x
Merry Christmas !
```

## 8.4 지역변수와 환경변수

# 환경변수와 지역변수

- 셀 변수
  - 환경변수와 지역변수 두 종류로 나눌 수 있다.
  - 환경 변수는 값이 자식 프로세스에게 상속되며 지역변수는 그렇지 않다.





## 환경변수와 지역변수 예

```
$ country=대한민국 city=서울
```

```
$ export country
```

```
$ echo $country $city
```

대한민국 서울

```
$ bash # 자식 셸 시작
```

```
$ echo $country $city
```

대한민국

```
$ ^D # 자식 셸 끝
```

```
$ echo $country $city
```

대한민국 서울

# 사전 정의 환경변수(predefined environment variable)

- 그 의미가 미리 정해진 환경변수들

| 이름         | 의미                  |
|------------|---------------------|
| \$USER     | 사용자 이름              |
| \$TERM     | 터미널 타입              |
| \$PATH     | 명령어를 검색할 디렉터리들의 리스트 |
| \$HOME     | 홈 디렉터리              |
| \$SHELL    | 로그인 셸의 경로명          |
| \$MAIL     | 메일 박스의 경로명          |
| \$HOSTNAME | 호스트 이름              |

\$ echo 홈 = \$HOME 사용자 = \$USER 셸 = \$SHELL

홈 = /user/faculty/chang 사용자 = chang 셸 = /bin/bash

\$ echo 터미널 = \$TERM 경로 리스트 = \$PATH

터미널 = xterm 경로 리스트 = /bin:/usr/bin:/usr/local/bin

# 사전 정의 지역 변수(predefined local variable)

| 이름        | 의미            |
|-----------|---------------|
| \$\$      | 셸의 프로세스 번호    |
| \$0       | 셸 스크립트 이름     |
| \$1 ~ \$9 | 명령줄 인수        |
| \$*       | 모든 명령줄 인수 리스트 |
| \$#       | 명령줄 인수의 개수    |

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

```
builtin.bash
```

```
echo 이 스크립트 이름: $0
```

```
echo 첫 번째 명령줄 인수: $1
```

```
echo 모든 명령줄 인수: $*
```

```
echo 이 스크립트를 실행하는 프로세스 번호: $$
```

```
$ builtin.bash hello shell
```

```
이 스크립트 이름: builtin.sh
```

```
첫 번째 명령줄 인수: hello
```

```
모든 명령줄 인수: hello shell
```

```
이 스크립트를 실행하는 프로세스 번호: 1259
```

## 8.5 Bash 셸 스크립트

# Bash 스크립트 작성 및 실행 과정

(1) 에디터를 사용하여 Bash 스크립트 파일을 작성한다.

```
#!/bin/bash
state.bash
echo -n 현재 시간:
date
echo 현재 사용자:
who
echo 시스템 현재 상황:
uptime
```

(2) chmod를 이용하여 실행 모드로 변경한다.

```
$ chmod +x state.bash
```

(3) 스크립트 이름을 타입핑하여 실행한다.

```
$ state.bash
```

# if 문

- if 문

```
if [조건식]
then
 명령들
fi
```

- 예

```
if [$# -eq 1]
then
 wc $1
fi
```

- if-then-else 구문

```
if [조건식]
then
 명령어리스트
else
 명령어리스트
fi
```

# if-then-else 문 예제: wc1.bash

```
#!/bin/bash
사용법: wc1.bash 파일
명령줄 인수 개수를 확인하고
 wc 명령어를 실행한다.
```

```
if [$# -eq 1]
then
 wc $1
else
 echo 사용법: $0 파일
fi
```

```
$ wc1.bash cs1.txt
38 318 2088 cs1.txt
```

## if-then-else 문 예제: count1.bash

```
#!/bin/bash
사용법: count1.bash [디렉터리]
대상 디렉터리 내의 파일과 서브디렉터리 개수를 프린트한다.
if [$# -eq 0]
then
 dir="."
else
 dir=$1
fi
echo -n $dir 내의 파일과 서브디렉터리 개수:
ls $dir | wc -l
```

```
$ count1.bash
```

```
. 내의 파일과 서브디렉터리 개수: 17
```



## 8.6 수식

# 비교 연산

- 비교 연산은 산술 비교 연산, 문자열 비교 연산

| 산술 비교 연산자   | 의미                          |
|-------------|-----------------------------|
| 정수1 -eq 정수2 | 두 정수가 같으면 참 아니면 거짓          |
| 정수1 -ne 정수2 | 두 정수가 다르면 참 아니면 거짓          |
| 정수1 -gt 정수2 | 정수1이 정수2보다 크면 참 아니면 거짓      |
| 정수1 -ge 정수2 | 정수1이 정수2보다 크거나 같으면 참 아니면 거짓 |
| 정수1 -lt 정수2 | 정수1이 정수2보다 작으면 참 아니면 거짓     |
| 정수1 -le 정수2 | 정수1이 정수2보다 작거나 같으면 참 아니면 거짓 |

# 문자열 비교 연산

| 문자열<br>비교 연산자 | 의미                     |
|---------------|------------------------|
| 문자열1 == 문자열2  | 두 문자열이 같으면 참<br>아니면 거짓 |
| 문자열1 != 문자열2  | 두 문자열이 다르면 참<br>아니면 거짓 |
| -n 문자열        | 문자열이 null이 아니면<br>참    |
| -z 문자열        | 문자열이 null이면 참          |

```
#!/bin/bash
사용법: reply.bash
계속 여부를 입력받아 프린트한다.
echo -n "계속 하겠습니까 ?"
read reply
read reply
if [$reply=="Y"] || [$reply=="y"]
then
 echo "계속합니다"
else
 echo "중지합니다"
fi

$ reply.bash
계속 하겠습니까 ?y
계속합니다
```

# 파일 관련 연산

| 파일 관련 연산자      | 의미                      |
|----------------|-------------------------|
| -a 파일<br>-e 파일 | 해당 파일이 존재하면 참           |
| -r 파일          | 사용자가 해당 파일을 읽을 수 있으면 참  |
| -w 파일          | 사용자가 해당 파일을 쓸 수 있으면 참   |
| -x 파일          | 사용자가 해당 파일을 실행할 수 있으면 참 |
| -O 파일          | 사용자가 해당 파일의 소유자이면 참     |
| -z 일           | 해당 파일의 크기가 0이면 참        |
| -f 파일          | 해당 파일이 일반 파일이면 참        |
| -d 파일          | 해당 파일이 디렉터리이면 참         |

## 파일 관련 연산: 예

```
if [-e $file]
then # $file이 존재하면
 wc $file
else # $file이 존재하지 않으면
 echo "오류 ! 파일 없음"
fi
```

```
if [-d $dir]
then
 echo -n $dir 내의 파일과 서브디
렉터리 개수:
 ls $dir | wc -l
else
 echo $dir\: 디렉터리 아님
fi
```

# 논리 연산

- 조건식에 부울 연산자 사용
  - ! 부정(negation)
  - && 논리곱(logical and)
  - || 논리합(logical or)

```
$file이 일반 파일이고 쓸수 있으면
if [-f $file] && [-w $file]
then
 uptime > $file
fi
```

```
if [! -e $file]
then # $file이 존재하지 않으면
 echo $file : 파일 없음
fi
```

```
if [! -d $file]
then # $dir이 디렉터리가 아니면
 echo $file : 디렉터리 아님
fi
```

# 산술 연산

- 산술 연산  
\$ a=2+3  
\$ echo \$a  
\$ a=`expr 2 + 3`
- let 명령어를 이용한 산술연산  
\$ let 변수=수식

```
$ let a=2*3
$ echo $a
6
$ let a=$a+2
$ echo $a
8
$ let a*=10
$ let b++
```

# 변수 타입 선언



- 변수 타입 선언: declare

```
$ declare -i a # a는 정수형 변수
$ a=12
$ a=a+1 # let 필요 없음
$ echo $a
$ a=12.3 # 오류 메시지
bash: 12.3: syntax error in
expr(error token is ".3")
$ declare -r b=23.4 # 읽기 전용
$ b=23.5 # 오류 메시지
bash: b: readonly variable
```

| 이름                           | 의미                         |
|------------------------------|----------------------------|
| <code>declare -r</code> 변수   | 읽기 전용 변수로 선언               |
| <code>declare -i</code> 변수   | 정수형 변수로 선언                 |
| <code>declare -a</code> 변수   | 배열 변수로 선언                  |
| <code>declare -f</code>      | 스크립트 안에서 정의된 모든 함수들을 보여준다. |
| <code>declare -f</code> 함수이름 | 해당 함수 이름을 보여준다.            |
| <code>declare -x</code> 변수   | 환경변수로 export               |



## 8.7 조건문

# Bash 제어구조

- 조건  
if
- 스위치  
case
- 반복  
for, while

# if 문

- if 문

```
if ((조건식))
then
 명령들
fi
```

- if-then-else 문

```
if ((조건식))
then
 명령들
else
 명령들
fi
```

# if 문

- 예

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

```
사용법: wc2.bash
```

```
명령줄 인수의 개수를 확인하고 wc 명령어를 실행한다.
```

```
if (($# == 1))
```

```
then
```

```
 wc $1
```

```
else
```

```
 echo 사용법: $0 파일
```

```
fi
```

# 수식 연산자

| 수식 연산자    | 의미                    |
|-----------|-----------------------|
| -         | 단일항 음수                |
| !         | 논리 부정                 |
| * / %     | 곱셈, 나눗셈, 나머지          |
| + -       | 덧셈, 뺄셈                |
| << >>     | 비트 좌이동, 비트 우이동        |
| <= >= < > | 관계 연산                 |
| == !=     | 동등, 비동등               |
| &&        | 논리합, 논리곱              |
| & ^       | 비트 and, 비트 xor, 비트 or |

# 중첩 if 문

- 중첩 if 문

```
if ((조건식))
then
 명령들
elif ((조건식))
then
 명령들
else
 명령들
fi
```

## 중첩 if 문: 예

```
#!/bin/bash
사용법: score1.bash
점수에 따라 학점을 결정하여 프린트
echo -n '점수 입력: '
read score
if (($score >= 90))
then
 echo A
elif (($score >= 80))
then
 echo B
elif (($score >= 70))
then
 echo C
else
 echo 노력 요함
```

\$score1.bash

점수 입력: 85

B

# case 문



```
case $변수 in
 패턴1) 명령들;;
 패턴2) 명령들;;
 ...
 *) 명령들;;
esac
```

```
#!/bin/bash
사용법: score2.bash
점수에 따라 학점을 결정하여 프
린트한다.
echo -n '점수 입력: '
read score
let grade=$score/10
case $grade in
 "10" | "9") echo A;;
 "8") echo B;;
 "7") echo C;;
 *) echo 노력 요함;;
esac
```



## 8.8 반복문

# for 문

- for 구문
  - 리스트의 각 값에 대해서 명령어들을 반복

```
for 변수명 in 리스트
do
 명령들
done
```

```
#!/bin/bash
사용법: invite.bash
저녁 초대 메일을 보낸다.
invitee=(lee kim choi)
for person in ${invitee[*]}
do
 echo "초대의 글 : 오늘 저녁
 식사 모임에 초대합니다." | \
 mail "${person}@gmail.com"
done
```

# 모든 명령줄 인수 처리

- 모든 명령줄 인수 처리

```
for file in $*
do
 echo $file
done
```

```
#!/bin/bash
사용법: perm1.bash 파일*
파일의 사용권한과 이름을 프린트한다.
if [$# -eq 0]
then
 echo 사용법: $0 파일*
 exit 1
fi
echo " 사용권한 파일"
for file in $*
do
 if [-f $file]
 then
 fileinfo=`ls -l $file`
 perm=`echo "$fileinfo"|cut -d' ' -f1`
 echo "$perm $file"
 fi
done
```

# while 문

- while 문

- 조건에 따라 명령어들을 반복적으로 실행

```
while ((조건식))
do
 명령들
done
```

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

```
사용법: power.bash
```

```
2의 1승부터 10승까지 프린트
```

```
let i=2
```

```
let j=1
```

```
while (($j <= 10))
```

```
do
```

```
 echo '2 ^' $j = $i
```

```
 let i*=2
```

```
 let j++
```

```
done
```

# menu.bash

```
#!/bin/bash
사용법: menu.bash
메뉴에 따라 해당 명령어를 실행한다.
```

echo 명령어 메뉴

```
cat << MENU
```

```
 d : 날짜 시간
 l : 현재 디렉터리 내용
 w : 사용자 보기
 q : 끝냄
```

```
MENU
```

```
stop=0
```

```
while (($stop == 0))
do
 echo -n '? '
 read reply
 case $reply in
 "d") date;;
 "l") ls;;
 "w") who;;
 "q") stop=1;;
 *) echo 잘못된 선택;;
 esac
done
```

# menu.bash

\$ menu.bash

명령어 메뉴

d : 날짜 시간

l : 현재 디렉터리 내용

w : 사용자 보기

q : 끝냄

? d

2012년 2월 23일 목요일 오후 07시 33분 27초

? q

# printf 문



\$ printf 포맷문자열 인수\*

특정 형식으로 형식화된 문자열을 출력한다.

포맷 문자열은 포맷 지정자 %를 사용하여 출력 형식을 지정하는 문자열이고, 인수는 포맷 지정자에 대응되는 값이다.

```
$ name="홍길동"
```

```
$ age=30
```

```
$ printf "이름: %s, 나이: %d\n" $name $age
```

```
이름: 홍길동, 나이: 30
```

## 8.9 고급 기능



# 디버깅

`$ bash -vx 스크립트 [명령줄 인수]`

```
$ bash -v menu.bash
#!/bin/bash
echo 명령어 메뉴
명령어 메뉴
cat << MENU
 d : 날짜 시간
 l : 현재 디렉터리 내용
 w : 사용자 보기
 q : 끝냄
MENU
 d : 날짜 시간
 l : 현재 디렉터리 내용
 w : 사용자 보기
 q : 끝냄
stop=0
```

```
while (($stop == 0))
do
 echo -n '? '
 read reply
 case $reply in
 "d") date;;
 "l") ls;;
 "w") who;;
 "q") stop=1;;
 *) echo 잘못된 선택;;
 esac
done
? d
2012년 ... 오후 07시 35분 05초
? q
```

# 함수

- 함수 정의  
함수이름()

```
{
 명령들
}
```

- 함수 호출  
함수이름 [매개변수]

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

```
사용법: lshad.bash
```

```
lshad() {
```

```
 echo "함수 시작, 매개변수 $1"
```

```
 date
```

```
 echo "디렉터리 $1 내의 처음 3개
 파일만 리스트"
```

```
 ls -l $1 | head -4
```

```
}
```

```
echo "안녕하세요"
```

```
lshad /tmp
```

# 함수

```
$!shead.bash
```

안녕하세요

함수 시작, 매개변수 /tmp

2022. 02. 23. (수) 17:43:27 KST

디렉터리 /tmp 내의 처음 3개 파일만 리스트

총 1184

```
-rw----- 1 chang faculty 11264 2009년 3월 28일 Ex01378
-rw----- 1 chang faculty 12288 2011년 5월 8일 Ex02004
-rw----- 1 root other 8192 2011년 5월 4일 Ex02504
```

## 디렉터리 내의 모든 파일 처리

- 디렉터리 내의 모든 파일 처리
  - 해당 디렉터리로 이동
  - for 문과 대표 문자 \*를 사용
  - 대표 문자 \*는 현재 디렉터리 내의 모든 파일 이름들로 대치

```
cd $dir
for file in *
do
 echo $file
done
```

## 디렉터리 내의 모든 파일 처리: count2.bash

```
#!/bin/bash
사용법: count2.bash [디렉터리]
대상 디렉터리 내의 파일, 서브디렉터리, 기타 개수를 세서 프린트
if [$# -eq 0]
then
 dir="."
else
 dir=$1
fi
if [! -d $dir]
then
 echo $0\: $dir 디렉터리 아님
 exit 1
fi
let fcount=0
let dcount=0
let others=0
```

## 디렉터리 내의 모든 파일 처리: 예

```
echo $dir\
cd $dir
for file in *
do
 if [-f $file]
 then
 let fcount++
 elif [-d $file]
 then
 let dcount++
 else
 let others++
 fi
done
echo 파일: $fcount 디렉터리: $dcount 기타: $others
```

# 자기 호출(recursion)

- 스크립트도 자기 자신을 호출 가능
- 어떤 디렉터리의 모든 하위 디렉터리에 대해 동일한 작업을 수행할 때 매우 유용함

```
#!/bin/bash
사용법 lssr.bash [디렉터리]
대상 디렉터리와 모든 하위 디렉터리 내에
있는 파일들의 크기를 리스트한다.
```

```
if [$# -eq 0]
then
 dir="."
else
 dir=$1
fi
```

```
if [! -d $dir]
then
 echo $0W: $dir 디렉터리 아님
 exit 1
fi
cd $dir
echo -e "\n $dir :"
ls -s
for x in *
do
 if [-d $x]
 then
 /home/chang/bash/lssr.bash $x
 fi
done
```

## 터미널에서 실행

- 터미널에서 while 혹은 for 문도 실행

```
$ for f in *
> do
> echo $f
> done
```

```
$ let i=2
$ let j=1
$ while (($j <= 10))
> do
> echo '2 ^' $j = $i
> let i*=2
> let j++
> done
2 ^ 1 = 2
2 ^ 2 = 4
...
2 ^ 10 = 1024
```



## 핵심 개념

- 단순 변수는 하나의 값(문자열)을 리스트 변수는 여러 개의 값(문자열)을 저장할 수 있다.
- 셸 변수는 크게 환경변수와 지역변수 두 종류로 나눌 수 있다. 환경 변수는 값이 자식 프로세스에게 상속되며 지역변수는 그렇지 않다.
- Bash 셸은 조건, 스위치, 반복 등을 위한 제어구조로 if, case, for, while 등의 문장을 제공한다.
- Bash 셸의 식은 비교 연산, 파일 관련 연산, 산술 연산 등을 할 수 있다.