시동파일과 쉘 스크립트 프로그래밍
(STARTUP FILE & SHELL SCRIPT PROGRAMMING)

안양대학교 컴퓨터공학과 하은용

시동파일과 쉘스크립트 프로그래밍

내용

- 시동 파일(Startup Files)
- 쉘 스크립트
- 쉘의 종류
- 쉘 프로그래밍

시동 파일

시동 파일 (startup files)이란

- 시스템, 혹은 응용프로그램의 기본 동작을 설정한 파일
  - Shells, Editors, Mailers, Debuggers, X Window System Clients, …
- 응용프로그램의 시작 초기에 시동파일을 읽고 초기 기능, 동작과 관련된 환경이 설정됨
- 일반적으로 일반사용자가 실행하는 프로그램들의 시동파일 이름은 '.*rc' 형태를 지님 (resource configuration)
- 쉘 시동파일은 스크립트 파일, 스크립트 프로그램의 일부로 명령어(예약어, 내장명령어, 외부명령어, 유틸리티)를 사용하여 환경을 설정하거나 초기 동작을 지정

주요 시동 파일

- .bashrc, .bash_profile: bash shell
- .cshrc, .login: c shell
- .emacs, .vimrc: editors
- /etc/rc.d: 시스템 부팅시 시스템과 서비스들을 설정
- .xinitrc: X Window System
- .kde/share/config: KDE
- .gnome: GNOME

쉘 스크립트 (1)

- 쉘 스크립트(shell script)란
  - 유닉스 기반 운영체제에서 쉘에 의해 해석되어 실행되는 일련의 명령들을 갖는 텍스트 파일
    - 쉘 스크립트를 구성하는 명령어: 예약어, 내장 명령어, 유닉스 명령어, 유틸리티
- 간단한 쉘 스크립트

```
# display the # of files and directories
# within current directory

echo –n "files and directories: "
ls | wc -l
```

'#'는 comments

  - 실행방법1: 현재 쉘

```
$ . num
$ source num
```

  - 실행방법2: 서브 쉘에 의해 실행

```
$ bash num
```

```
$ chmod u+x num
$ num
```

쉘 스크립트 (2)

해석 쉘의 명시

- 특별한 경우를 제외하고 대부분의 쉘 스크립트에는 스크립트를 해석하여 실행하는 쉘을 명시

```
#!/bin/bash
# display the # of files and directories
# within current directory

echo –n "files and directories: "
ls | wc -l
```

'#!'는 magic number
첫 라인에서만 유효

- 명시 하지 않는 경우에는 사용자 기본 쉘에 의해 실행
  - 사용자 기본 쉘의 확인: $ echo $SHELL
  - 쉘의 종류에 따라 쉘 내장 명령어가 다르므로 명시하지 않으면 오동작 가능성 존재
- 특별한 경우에는 정해진 쉘에 의해 실행
  - ex) ~/.bash_profile, ~/.bashrc

.bash_profile & .bashrc

구동 파일의 기본 내용

.bash_profile

```
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi

# User specific environment and startup programs
```

.bashrc

```
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi
# User specific aliases and functions
```

예약어: if, then, fi
내장 명령: [, .
제어연산자: ;

제어연산자란 명령 제어 기능을 수행하는 토큰
제어연산자:
||, &, &&, ;, ;;, ( ), |, |&, <newline>

대화형 로그인 쉘

대화형 로그인 쉘(Interactive login shell)

- 대화형이란 사용자가 명령프롬프트에 명령을 입력하여 그 결과를 볼 수 있다는 의미
- 로그인 쉘은 사용자명(UID)과 패스워드를 사용한 인증을 통해 쉘이 실행
  - console 로그인, 원격로그인, bash -l (bash --login)

시동파일

- 대화형 로그인 쉘은 다음 시동 파일을 순서대로 읽어 설정 및 실행
  - /etc/profile : 통상적으로 시스템 전역적인 설정 및 프로그램 초기화, 다음 시동파일 불러서 읽고 실행
    - /etc/inputrc : 입력 키 설정
    - /etc/profile.d/*.sh: 개별 프로그램을 위한 시스템 전역 설정 파일
  - ~/.bash_profile, ~/.bash_login or ~/.profile 중 먼저 존재하는 파일
  - ~/.bash_logout 로그아웃시

대화형 비로그인 쉘

- 대화형 비로그인 쉘(interactive non-login shell)
  - 대화형 쉘이지만 인증이 필요 없는 쉘
    - X Window의 아이콘이나 메뉴를 사용하여 터미널을 실행할 때
    - 인자 없이 'bash' 명령어로 자식 쉘을 생성하였을 때
- 시동파일
  - 이러한 쉘의 시동 파일: ~/.bashrc
    - 통상적으로 ~/.bash_profile에서 다음과 같이 참조됨

```
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi
```

```
if [ -f ~/.bashrc ]
then
    . ~/.bashrc
fi
```

```
if list
then
    commands
[elif list
then
    commands]
[else
    commands]
fi
```

    - /etc/bashrc를 참조

비대화형 쉘

- 비대화형 쉘(Non-interactive shell)
  - 쉘 스크립트가 실행될 때 쉘 스크립트의 명령을 해석하는 쉘, 혹은 즉 옵션이 아닌 인자와 실행될 때, 쉘이 '-c' 옵션으로 실행될 때
  - PS1 환경변수가 unset, BASH_ENV 환경 변수에 지정된 시동파일을 읽고 설정 및 실행
- 예제

```
$ cd
$ cat > greetings
GREETINGS="Hello $USER!"
[Ctrl+D]
$ export BASH_ENV=~/greetings
$ cat > hello
#!/bin/bash
echo $GREETINGS
[Ctrl+D]
$ chmod u+x hello
$ hello
Hello mylinux!
```

```
$ . hello

$ GREETINGS="Hello Linux"
$ source hello
Hello Linux
$ bash hello
Hello mylinux!
$ ./hello
Hello mylinux!
```

대화형 쉘이 비대화형 쉘과 다른 점

- 시동파일을 읽음, 작업제어가 가능
- PS1, PS2가 설정됨, 명령프롬프트에서 명령을 읽고 실행됨
- 현재 실행중인 쉘이 대화형 쉘인지 확인하는 방법
  - PS1가 null이 아니거나, '$-'의 값에 문자 'i'가 포함되어 있다면 대화형쉘
- ignoreeof 설정 유효
- 명령 히스토리, 히스토리 확장 사용 가능, HISTFILE 유효
- alias 확장이 활성화됨, 주기적으로 mail을 체크
- 구문오류나 exec의 실행시 오류가 발생하더라도 쉘이 종료되지 않음
- TMOUT 변수에 시간 길이가 지정되었을 때 자동 종료
- 표준입력 장치, 표준출력 장치 둘 다 터미널에 연결되어 있음

쉘 스크립트에서 인자

- 인자(arguments) 지정하기
  - $#: 인자의 수, $*: 전체 인자, $0: 프로그램명, $1~$9: n번째 인자
- 인자를 가지는 스크립트 예제

```
#!/bin/bash

# display the # of files and directories
echo –n "files and directories: "
ls $1 | wc –l

exit 0
```

  - 위 예제에서 인자가 하나도 주어지지 않았을 경우에는 오류가 발생
  - 오른쪽 예에서 존재하지 않는 디렉터리를 인자로 지정한다면?

```
#!/bin/bash
# display the # of files and directories

if [ $# -eq 0 ]; then
    echo –n "files and directories: "
    ls | wc –l
elif [ $# -eq 1 ]; then
    echo –n "files and directories: "
    ls $1 | wc –l
else
    echo "Usage : $0 directory" 1>&2
    exit 1
fi
exit 0
```

키워드와 내장 명령어

- 키워드(예약어)
  - 쉘에게 특별한 의미를 갖는 단어
  - !, case, do, done, elif, else, esac, fi, for, function, if, in, select, then, time, until, while, {, }, time, [[,]]
- 내장 명령어
  - 실행시 별도의 프로세스를 생성하지 않고 쉘의 일부로 수행되는 명령어
  - :, source, ., alias, unalias, bg, fg, jobs, break, cd, continue, declare, echo, eval, exec, exit, export, false, getopts, hash, help, history, kill, let, local, printf, pwd, dirs, pushd, popd, read, return, readonly, set, unset, shift, shopt, test, [, times, true, type, umask

단순 명령

- 단순 명령(simple commands)
  - 제어 연산자로 끝나는 명령
  - 제어 연산자: ||, &, &&, ;, ;;, ( ), |, |&, <newline>
  - 종료상태(exit status)는 단순명령 반환값

종료 상태

- 종료 상태(exit status)란
  - 실행된 명령어의 종료상태는 명령어를 실행한(호출한) 측에 반환되는 값을 의미
  - 0~ 255 사이의 값을 가짐, 127 이상의 값은 특별히 취급됨
  - 직전에 수행 완료된 명령의 종료 상태 확인 명령:
    - $ echo $?
- 쉘이 인지하는 수행성공과 수행실패
  - 종료 상태가 0이면 명령수행 성공 (true), 0이 아니면 명령수행 실패 (false)
  - 특정 시그널 N에 의해 종료된 경우의 종료 상태: 128+N
  - 명령어가 없는 경우의 종료 상태: 127
    - 명령어가 없는 경우에도 쉘은 자식 프로세스를 생성하고 실행을 시도
  - 명령어는 존재하나 실행 불가능한 상태일 경우 종료 상태: 126

명령 리스트

- ‘;’, ‘&’, <newline>으로 끝나고
- ‘;’, ‘&’, ‘&&’, ‘||’로 연결되는 하나 이상의 명령
  - ‘&’ 앞의 명령어는 서브 쉘에서 백그라운드로 수행, 종료상태는 0
  - ‘;’로 구분되는 명령어는 연속으로 수행되며 쉘은 명령어가 수행완료 되기를 기다림.
  - 종료상태는 ‘;’로 연결된 명령 중 마지막 명령의 종료상태
- command1 || command2
  - command1이 거짓이면(종료상태가 0이 아니면) command2 수행
  - command1이 참이면(종료상태가 0이면) command2 수행하지 않음
- command1 && command2
  - command1이 참이면 command2 수행
  - command1이 거짓이면 command2 수행하지 않음
- command1 ; command2
  - command1과 command2를 무조건 수행 시도

복합문 (1)

- (list)
  - list는 서브쉘에서 실행됨
- { list; }
  - list는 현재쉘에서 실행됨
- ((산술표현식))
  - 기술된 산술표현식이 평가됨
  - 평가된 표현식의 값이 0이 아니면 종료상태는 0, 그렇지 않으면 종료상태는 1.
  - let "표현식" 명령과 동일
- [[조건표현식]]
  - 기술된 조건표현식이 평가됨
  - 이는 명령어 test나 [를 사용하여 'test 조건표현식', 혹은 '[조건표현식]' 형태의 확장된 구문

복합문 (2)

- If 문
  - if list; then list; [elif list; then list;] ... [else list;] fi
- Case 문
  - case word in [pattern[| pattern]) list ;;] ... esac
- Loop 문
  - for name [in word ...] ; do list ; done
  - while list; do list; done
  - until list; do list; done
  - select name [in word] ; do list ; done

표현식 – 산술 표현식

- 산술 표현식 연산자
  - C언어에서 사용 가능한 산술연산자는 모두 사용 가능
  - 연산자들의 우선순위, 결합법칙 모두 C언어와 동일
- 사용 가능한 연산자들의 종류 (우선순위에 따라)
  - id++, id--, ++id, --id
  - -, +: 단항 연산자
  - !, ~: 논리부정, 비트반전
  - **
  - *, /, %: 곱하기, 나누기, 나머지
  - +, -: 이항연산자 더하기, 빼기
  - <<, >>: 비트연산자
  - <=, >=, <, >: 비교
  - ==, !=: 비교
  - &: 비트 AND
  - ^: 비트 XOR
  - |: 비트 OR
  - &&: 논리 AND
  - ||: 논리 OR
  - ?: 3항 연산자
  - =, *=, /=, %=, +=, -=, <<=, >>=, &=, ^=, |=: 대입
  - ,: 나열

표현식 – 조건 표현식

조건 표현식이란

- 파일의 속성이나 문자열의 비교, 산술 비교를 검사하기 위해
  - [[조건표현식]]
  - test 조건표현식
  - [조건표현식] 에 의해 사용됨
- 다음 조건 표현식에서
  - file이 /dev/fd/N의 형태라면 파일디스크립터 N인 파일이 검사되며,
  - /dev/stdin, /dev/stdout, /dev/stderr이면 파일디스크립터가 각각 0, 1, 2인 파일이 검사됨

파일 속성 검사 (1)

- -a file : 파일이 존재하면 true
- -b file : 파일이 존재하고 블록 장치 파일이면 true
- -c file : 파일이 존재하고 문자 장치 파일이면 true
- -d file : 파일이 존재하고 디렉터리이면 true
- -e file : 파일이 존재하면 true
- -f file : 파일이 존재하고 일반 파일이면 true
- -g file : 파일이 존재하고 set-group-id가 설정되어 있으면 true
- -h file : 파일이 존재하고 심볼릭 링크이면 true
- -k file : 파일이 존재하고 sticky bit가 설정되어 있으면 true
- -p file : 파일이 존재하고 이름 붙은 파이프이면 true
- -r file : 파일이 존재하고 읽기 가능하면 true

파일 속성 검사 (2)

- -s file : 파일이 존재하고 크기가 0보다 크면 true
- -t fd : 파일 디스크립터 fd가 열려있고 terminal이면 true
- -u file : 파일이 존재하고 set-user-id가 설정되어 있으면 true
- -w file : 파일이 존재하고 쓰기 가능하면 true
- -x file : 파일이 존재하고 실행 가능하면 true
- -O file : 파일이 존재하고 유효사용자의 소유이면 true
- -G file : 파일이 존재하고 유효그룹의 소유이면 true
- -L file : 파일이 존재하고 심볼릭 링크이면 true
- -S file : 파일이 존재하고 소켓이면 true
- -N file : 파일이 존재하고 최근 접근(읽기) 이후 변경되었으면 true

파일 비교

- file1 -nt file2
  - file1이 file2보다 더 최근에 변경되었거나, file1이 존재하고 file2가 존재하지 않는 경우 true
- file1 -ot file2
  - file1이 file2보다 더 이전에 변경되었거나, file2가 존재하고 file1이 존재하지 않는 경우 true
- file1 -ef file2
  - file1과 file2가 같을 경우 (inode number를 기준)

문자열 비교

- -z str : str의 길이가 0이면 true
- -n str : str의 길이가 0이 아니면 true
- str1 == str2
- str1 = str2 : 두 문자열이 같다면 true
- str1 != str2 : 두 문자열이 같지 않다면 true
- str1 < str2 : str1이 str2보다 빠른 순서일 경우 true
- str1 > str2 : str1이 str2보다 나중 순서일 경우 true

산술 비교

- a1 -eq a2 : a1이 a2와 같은 경우 true
- a1 -ne a2 : a1이 a2와 같지 않을 때 true
- a1 -lt a2 : a1이 a2보다 작을 때 true
- a1 -le a2 : a1이 a2보다 작거나 같을 때 true
- a1 -gt a2 : a1이 a2보다 클 때 true
- a1 -ge a2 : a1이 a2보다 크거나 같을 때 true

선택문 – if (1)

- if 문장의 개요
  - 조건에 따라 다른 동작을 지정할 수 있는 쉘 명령 구문
    - 명령 프롬프트에서도 사용 가능하지만, 통상적으로 쉘스크립트 프로그래밍에서 사용
- 구문 형태: if list1; then list2; fi
  - if list1은 list1을 실행하고 종료상태가 0이면 list2를 실행
    - list1은 어떤 명령어도 가능하지만 일반적으로 숫자나 문자열의 검사/비교가 주로 사용됨

선택문 – if (2)

완전한 구문 형태

```
if list
then list
[elif list
then list]
[else list]
fi
```

```
if list
then
    list
[elif list
then
    list]
[else
    list]
fi
```

```
if list; then list;
[elif list; then list]
[else list;] fi
```

- ‘if list’가 실행되고 종료 상태가 0(true)이면 then list를 수행, 종료 상태가 1(false)이면 ‘elif list’ 수행, …

선택문 – if (3)

예제

```
$ if id mylinux &> /dev/null
> then grep ^mylinux /etc/passwd | cut -d: -f5
> else echo "mylinux does not exist"
> fi
Linux Programmer
$ if id mylinux &> /dev/null; then grep ^nousr /etc/passwd |
cut -d: -f5; else echo “nouser does not exist"; fi
nouser does not exist
$ if grep hello file1
> then echo Found
> else echo Not Found
> fi
Found
```

선택문 – case (1)

- case word in [pattern[| pattern]) list ;;] ... esac
  - word와 일치하는 pattern의 case에 상응하는 list를 수행
  - '|'을 사용하면 하나의 case에 여러 개의 pattern을 지정할 수 있음

```
#!/bin/bash
aflag=0
bflag=0

if [ $# -ne 1 ]; then
    echo "Usage: $(basename $0) arg" >&2
    exit 1
fi
case $1 in
    -a) aflag=1
        ;;
    -b) bflag=1
        ;;
    *)  echo "Usage: $(basename $0) [-a][-b]" >&2
        exit 1
esac
echo "aflag=$aflag, bflag=$bflag"
```

검사(test) 명령 (1)

- test 명령
  - 파일 타입을 검사, 문자열이나 값을 비교하는 내장 명령어
- 명령 형식

```
test 표현식
[ 표현식 ]
```

- 예제1

```
$ test -f .bashrc && echo file exist || file does not exist
$ [ -f .bashrc ] && echo file exist || file does not exist
$ test 3 -gt 4 && echo True || echo False
$ [ "abc" != "def" ]; echo $?
$ test -d "$HOME" && echo $HOME is a directory || echo $HOME is not directory
```

검사(test) 명령 (2)

예제2

```
#!/bin/bash

if [ $# -eq 0 ]; then
    echo "Usage: $0 userid" >&2
    exit 1
fi

w="$(who | grep $1)"
if [ $w -eq 0 ]; then
    echo "$1 is not logged in"
else
    echo "$1 is logged in"
fi

exit 0
```

루프(Loop) - for 문 (1)

- for name [in word ...] ; do list ; done
  - 'in' 뒤에 나열된 'word ...'에서 각각 word를 name(변수)의 값으로 하여 list가 한 번씩 수행됨
  - 'in word ...'가 지정되지 않는다면 'in $@'로 대치됨.
  - 종료상태는 마지막 수행된 명령의 종료상태
- 예제1

```
#!/bin/bash
for i; do
    cp "$i" "$i".bak
done
exit 0
```

```
$ ls src
man.c  main.o  hello.c  hello.o common.c
common.o
$ ./backup src/*.c
$ ls src
man.c  main.c.bak  main.o  hello.c
hello.c.bak  hello.o  common.c
common.c.bak  common.o
```

루프(Loop) - for 문 (2)

예제2

```
#!/bin/bash
for i; do
    if [ ! -d "$i" ]; then
        echo "$i is not a directory" >&2
        exit 1
    fi
    for f in `ls $i`; do
        file $i/$f | grep "ASCII\|script"
    done
done

exit 0
```

루프(Loop) - while 문 (1)

- while list1; do list2; done
  - list1을 수행하고 종료상태가 0이면 list2를 수행
  - list1의 종료상태가 0이 아닐 때까지 list2를 반복 수행
  - 종료상태는 list2에서 마지막으로 수행된 명령의 종료상태
- 예제1

```
#!/bin/bash
if [ $TERM != "xterm" ]; then
    echo "current terminal is not xterminal"
    exit 1
fi
i=0
while [ $i -lt 4 ]; do
    gnome-terminal &
    let "i += 1"
done
```

루프(Loop) - while 문 (2)

- 예제2

```
#!/bin/bash
# 입력한 점수 평균계산

SCORE="0"
AVERAGE="0"
SUM="0"
NUM="0"

while true; do
    echo -n "Enter your score [0-100%] "
    echo -n "('q' for quit): "
    read SCORE;
    if (("$SCORE" < "0")) ||
       (("$SCORE" > "100")); then
        echo "Be serious. Common, try again: “
    elif [ "$SCORE" == "q" ]; then
        echo "Average rating: $AVERAGE%."
        break
    else
        SUM=$(($SUM + $SCORE))
        NUM=$(($NUM + 1))
        AVERAGE=$(($SUM / $NUM))
    fi
done
echo "Exiting."
```

break, continue

- **break**
  - 루프 문 내에서 정상종료 전에 루프를 빠져나가는 명령
  - 인자로 숫자를 지정한다면 지정한 숫자 레벨만을 빠져나감. 숫자는 반드시 1보다 커야 하며, break문을 둘러싸고 있는 루프 레벨보다 크다면 전체를 빠져나감
- **continue**
  - 다음 루프를 수행하는데 사용되는 명령

select 명령으로 메뉴 만들기 (1)

select 구문 형식

- select name [in word ...] ; do list ; done
- select 명령 구문을 이용하면 메뉴를 손쉽게 생성 가능
- 각각의 word에 대해 “숫자) word” 형태의 메뉴가 출력되고 PS3에 지정된 프롬프트가 출력되어 ‘숫자’가 입력되기를 기다림
- ‘숫자’를 입력했을 때 해당 word는 name에 저장되고 list가 수행됨
- 'in word ...'가 지정되지 않았다면 'in $@'가 대신함
- 범위 내에 없는 숫자를 입력하면 목록을 다시 출력하고 다시 숫자를 입력하기를 기다림

select 명령으로 메뉴 만들기 (2)

예제1: private

```
#!/bin/bash
echo -n "This script can make any of the "
echo "files in this directory private."
echo -n "Enter the number of the file "
echo "you want to protect:"

select FILENAME in *;
do
    echo -n "You picked $FILENAME, "
    echo "it is now only accessible to you."
    chmod go-rwx "$FILENAME"
done
```

```
$ ls
file1  file2  private
$ ./private
1) file1  2) file2  3) private
#? 1
You picked file1, it is now only accessible
to you.
#?
```

select 명령으로 메뉴 만들기 (3)

예제2

```
#!/bin/bash

PS3="Your choice: "
echo -n "This script can make any of the "
echo "files in this directory private."
echo -n "Enter the number of the file "
echo "you want to protect:"

select FILENAME in *;
do
    if [ $REPLY == "q" ]; then
        break
    fi
    echo -n "You picked $FILENAME, "
    echo it is now only accessible to you."
    chmod go-rwx "$FILENAME"
done
```

```
$ ls
file1  file2  private
$ ./private
1) file1  2) file2  3) private
Your choice: q
$
```

함수 (1)

함수 정의 형식

- function function_name { list; }
- function_name() { list; }
  - 내장 명령 'function'을 사용하지 않는다면 함수명 다음에 괄호가 반드시 지정되어야 함
  - 종료상태는 마지막 명령의 종료상태

주의 사항

- 중괄호 전후에 반드시 빈 공백이 와야 함
- 특정 변수를 함수 내에서만 사용하려면 변수를 local 명령으로 정의해야 함
- 함수 인자 전달을 위해 위치 매개변수를 사용

함수 (2)

예제

```
#!/bin/bash
echo "Positional parameter (script): $1"
test ()
{
    echo "Positional parameter (function): $1"
    RETURN_VALUE=$?
    echo "The exit code: $RETURN_VALUE."
}
test other_param
```

내장 명령어 – read (1)

- read 명령의 특징과 기능
  - 쉘 내장 명령
  - 표준입력으로부터 한 줄을 읽어 명령인자로 지정된 변수에 저장
  - 한 줄에서 첫 번째 단어는 첫 번째 변수에 저장, 두 번째 단어는 두 번째 변수, ... 나머지 단어는 맨 마지막 변수에 저장됨
- 명령 형식
  - read [-s|-n*M*|-p *string*] [*var* ...]

```
$ read myname
Chulsoo Kim
$ echo $myname
Chulsoo Kim
$ read aname bname
Chulsoo Kim
$ echo "$aname:$bname"
Chulsoo:Kim
```

쉘 내장 명령어 – read (2)

- -s: 입력 문자열을 에코하지 않음

```
$ read -s var1 var2
$ echo $var1
This
$ echo $var2
is a built-in command
```

- -nN: N개의 입력만 받음, N개의 입력이 들어오면 엔터키 입력없이 곧바로 수행 완료됨

```
$ read -n5 var
linux
$ echo $var
linux
```

- -p string: string을 표준 출력하고 입력 값을 기다림

```
$ read -p "name: " name
name: mylinux
$ echo $name
mylinux
```

쉘 내장 명령어 – read (3)

- REPLY 쉘변수
  - 명령인자로 변수명을 생략하면 입력값은 REPLY라는 쉘변수에 저장됨

```
$ read
Hello
$ echo $REPLY
Hello
```

- 파일 리다이렉션과 같이 사용하기
  - 파일을 리다이렉션할 경우 파일의 첫 번째 줄만 변수의 값으로 지정됨

```
$ cat file1
Hello World
Good bye!
$ read var1 var2 < file1
$ echo $var1
Hello
$ echo $var2
World
```

내장 명령어 - *pushd, popd, dirs*

- **pushd, popd, dirs 명령 용도**
  - 몇몇 특정 디렉터리를 오고가며 작업할 경우 이동을 용이하게 함
  - 모두 쉘 내장명령(built-in command)
- **pushd의 기능**
  - 현재 작업디렉터리를 명령 인자로 지정한 디렉터리 경로로 변경, 디렉터리 스택의 top에 push
  - 동시에 디렉터리 스택의 내용을 표준출력
- **popd의 기능**
  - 디렉터리 스택 top의 항목 제거, 제거한 후 현재 작업디렉터리를 제거된 뒤의 디렉터리 스택 top으로 변경
  - 동시에 디렉터리 스택의 내용을 표준 출력
- **dirs의 기능**
  - 현재의 디렉터리 스택 내용을 표준 출력
  - 디렉터리 스택의 내용은 DIRSTACK 환경변수에도 저장됨

내장 명령어 - *umask*

umask의 기능

- 파일이나 디렉터리가 생성될 때 정해지는 기본적인 접근권한 제어
- 'umask' 명령은 현재 설정된 umask 값을 출력
- 명령인자로 숫자가 지정되어 있으면 그 숫자는 8진수로 해석됨

umask의 설정

- 생성되는 파일의 접근 권한은 666에 umask의 값을 뺀 값이 설정됨
- 생성되는 디렉터리의 접근권한은 777에 umask의 값을 뺀 값이 설정됨

umask	디렉터리	파일
002	775	664
007	770	66-
020	757	646
070	705	604
022	755	644

요약

- Bash 쉘 시동파일
  - ~/.bash_profile, ~/.bashrc
- 쉘 스크립트
- 쉘의 종류에 따른 시동파일
  - 대화형 로그인 쉘, 대화형 비로그인 쉘, 비대화형 쉘
- 쉘 명령 문법: 단순명령, 명령나열, 복합명령
- 종료상태, 검사명령
- if, while, until, for, case, 함수, 내장명령어