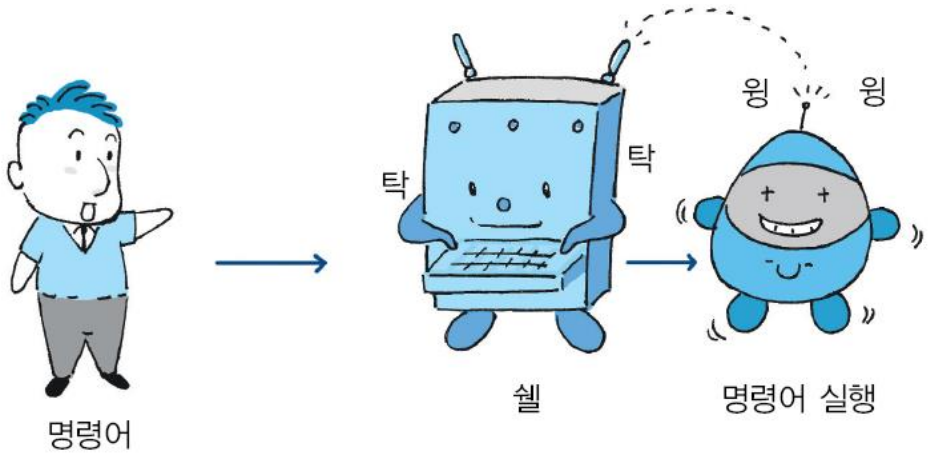
제5장 쉘과 명령어 사용

숙명여대 창병모

5.1 쉘 소개

쉘(Shell)이란 무엇인가?

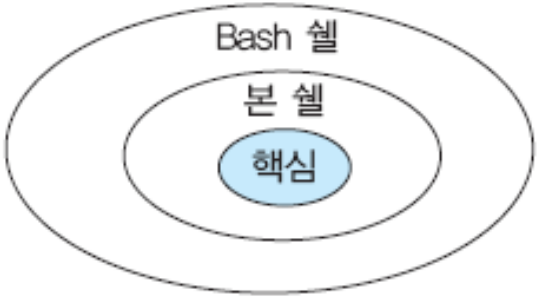
- 쉘의 역할
  - 쉘은 사용자와 운영체제 사이에 창구 역할을 하는 소프트웨어
  - 명령어 처리기(command processor)
  - 사용자로부터 명령어를 입력받아 이를 처리한다



쉘의 종류

- 유닉스/리눅스에서 사용 가능한 쉘의 종류

쉘의 종류	쉘 실행 파일
본 쉘	/bin/sh
콘 쉘	/bin/ksh
C 쉘	/bin/csh
Bash	/bin/bash
tcsh	/bin/tcsh



쉘의 종류

- 본 쉘(Bourne shell)
  - 벨연구소의 스티븐 본(Stephen Bourne)에 의해 개발됨
  - 유닉스에서 기본 쉘로 사용됨
- 콘 쉘(Korn shell)
  - 1980년대에는 역시 벨연구소에서 본 쉘을 확장해서 만듬.
- Bash(Bourne again shell)
  - GNU에서 본 쉘을 확장하여 개발한 쉘
  - 리눅스 및 맥 OS X에서 기본 쉘로 사용되면서 널리 보급됨
  - Bash 명령어의 구문은 본 쉘 명령어 구문을 확장함
- C 쉘(C shell)
  - 버클리대학의 빌 조이(Bill Joy)
  - 쉘의 핵심 기능 위에 C 언어의 특징을 많이 포함함
  - BSD 계열의 유닉스에서 많이 사용됨
  - 최근에 이를 개선한 tcsh이 개발됨어 되어 사용됨

로그인 쉘(login shell)

- 로그인 하면 자동으로 실행되는 쉘
- 보통 시스템관리자가 계정을 만들 때 로그인 쉘 지정

/etc/passwd

```
root:x:0:1:Super-User:/:/bin/bash
...
chang:x:109:101:Byeong-Mo Chang:/user/faculty/chang:/bin/bash
```

로그인 쉘 변경

- 쉘 변경

```
$ csh
%
...
% exit
$
```

- 로그인 쉘 변경

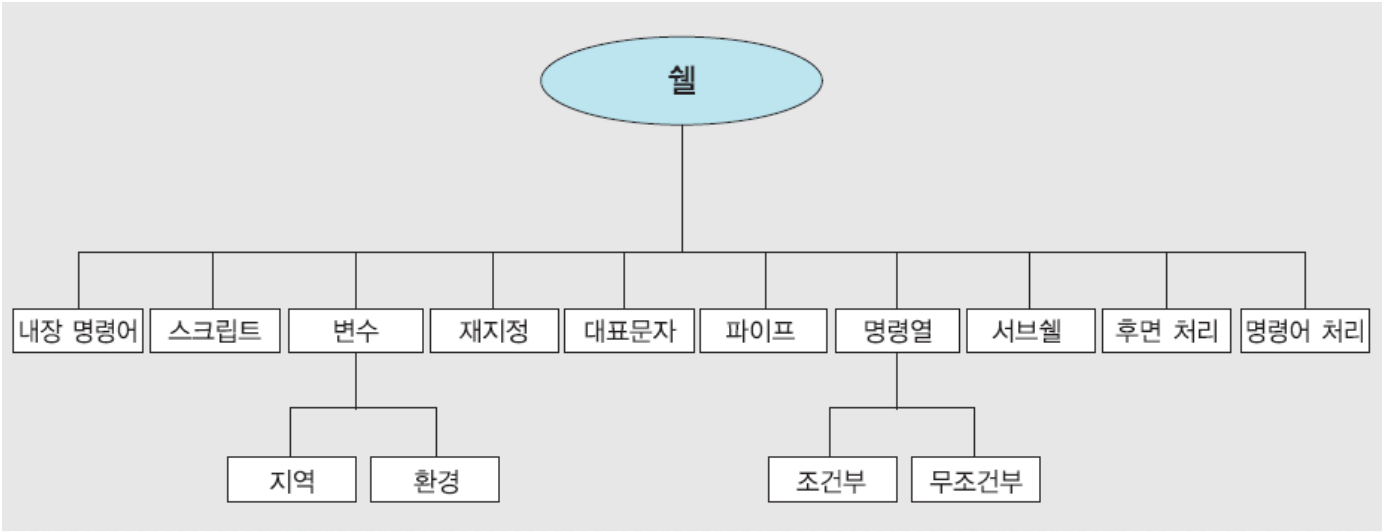
```
$ chsh
Changing login shell for chang
Old shell : /bin/sh
New shell : /bin/csh
$ logout

login : chang
passwd:
%
```

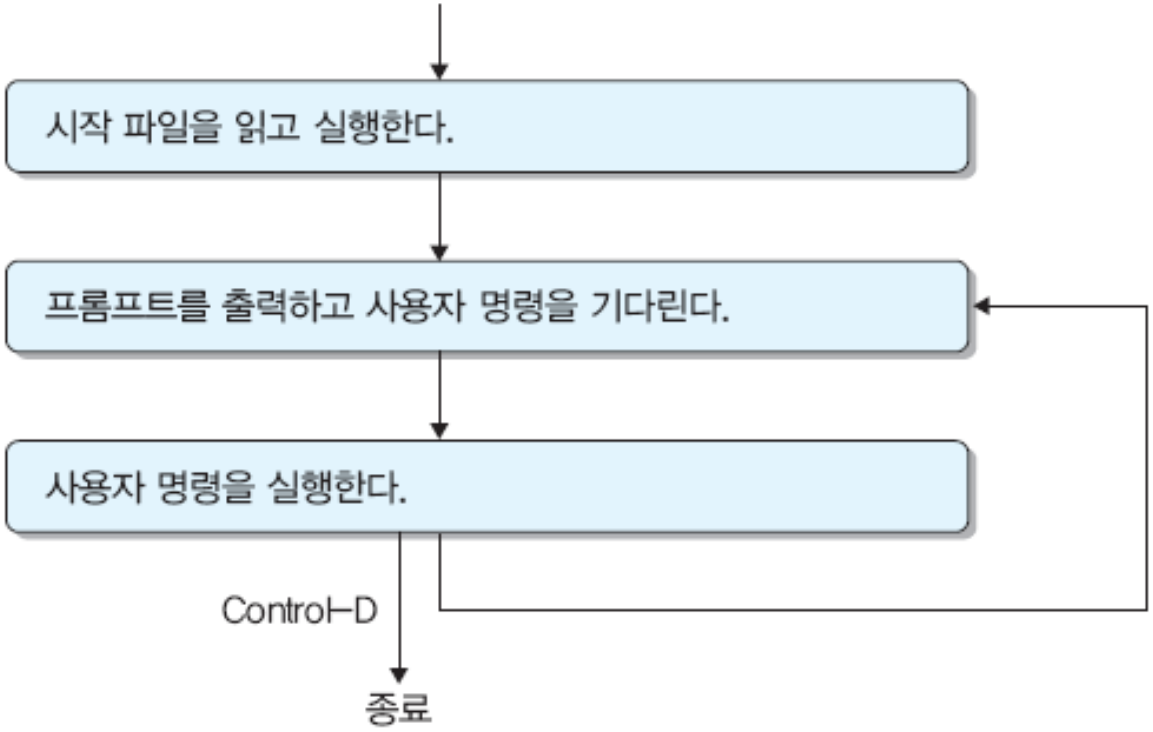
5.2 쉘의 기능

쉘의 주요 기능

- 명령어 처리
  - 사용자가 입력한 명령을 해석하고 적절한 프로그램을 실행
- 시작 파일
  - 로그인할 때 실행되어 사용자별로 맞춤형 사용 환경 설정
- 스크립트
  - 쉘 자체 내의 프로그래밍 기능



쉘의 실행 절차


시작 파일을 읽고 실행한다.
프롬프트를 출력하고 사용자 명령을 기다린다.
사용자 명령을 실행한다.
Control-D
종료

쉘의 환경 변수

- 환경변수 설정법

> $ 환경변수명=문자열
>
> 환경변수의 값을 문자열로 설정한다.

- 예

```
$ TERM=xterm
$ echo $TERM
xterm
```

쉘의 환경 변수

- 환경변수 보기

```
$ env
TERM=xterm
SHELL=/bin/sh
GROUP=cs
USER=chang
HOME=/home/chang
PATH=/usr/local/bin:/usr/bin: ...
...
```

- 사용자 정의 환경 변수

```
$ MESSAGE=hello
$ export MESSAGE
```

쉘의 시작 파일(start-up file)

- 시작 파일
  - 쉘마다 시작될 때 자동으로 실행되는 고유의 시작 파일
  - 주로 사용자 환경을 설정하는 역할을 하며
  - 환경설정을 위해서 환경변수에 적절한 값을 설정한다.
- 시스템 시작 파일
  - 시스템의 모든 사용자에게 적용되는 공통적인 설정
  - 환경변수 설정, 명령어 경로 설정, 환영 메시지 출력, ...
- 사용자 시작 파일
  - 사용자 홈 디렉터리에 있으며 각 사용자에게 적용되는 설정
  - 환경변수 설정, 프롬프트 설정, 명령어 경로 설정, 명령어 이명 설정, ...

시작 파일(start-up file)

쉘의 종류	시작파일 종류	시작파일 이름	실행 시기
본 쉘	시스템 시작파일	/etc/profile	로그인
	사용자 시작파일	~/.profile	로그인
Bash 쉘	시스템 시작파일	/etc/profile	로그인
	사용자 시작파일	~/.bash_profile	로그인
	사용자 시작파일	~/.bashrc	로그인, 서브쉘
	시스템 시작파일	/etc/bashrc	로그인
C 쉘	시스템 시작파일	/etc/.login	로그인
	사용자 시작파일	~/.login	로그인
	사용자 시작파일	~/.cshrc	로그인, 서브쉘
	사용자 시작파일	~/.logout	로그아웃

시작 파일 예

- .profile

```
PATH=$PATH:/usr/local/bin:/etc
TERM=vt100
export PATH TERM
stty erase ^
```

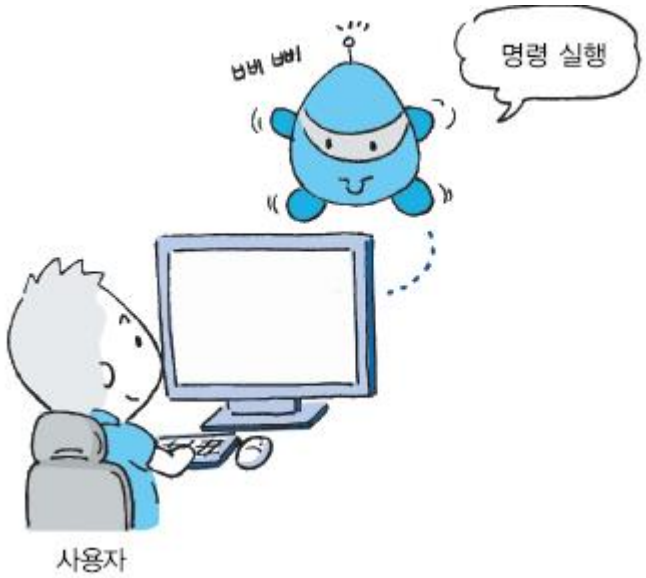
- 시작 파일 바로 적용

```
$ . .profile
```

5.3 전면 처리와 후면 처리

전면 처리 vs 후면처리

- 전면 처리
  - 입력된 명령어를 전면에서 실행하고 쉘은 명령어 실행이 끝날 때까지 기다린다.
  - $ 명령어
- 후면 처리
  - 명령어를 후면에서 실행하고 전면에서는 다른 작업을 실행하여 동시에 여러 작업을 수행할 수 있다.
  - $ 명령어 &



후면 처리 예

- $ (sleep 100; echo done) &
  [1] 8320
- $ find . -name test.c -print &
  [2] 8325

후면 작업 확인

- 사용법

```
$ jobs [%작업번호]
```

후면에서 실행되고 있는 작업들을 리스트 한다. 작업 번호를 명시하면 해당 작업만 리스트 한다.

- 예

```
$ jobs
[1] + Running ( sleep 100; echo done )
[2] - Running find . -name test.c -print
$ jobs %1
[1] + Running ( sleep 100; echo done )
```

후면 작업을 전면 작업으로 전환

- 사용법

```
$ fg %작업번호
```

작업번호에 해당하는 후면 작업을 전면 작업으로 전환시킨다.

- 예

```
$ (sleep 100; echo DONE) &
[1] 10067
$ fg %1
( sleep 100; echo DONE )
```

5.4 입출력 재지정

출력 재지정(output redirection)

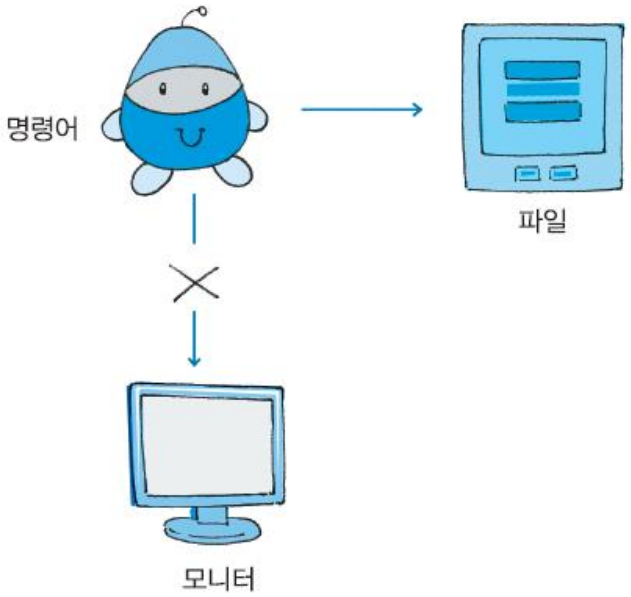
- 사용법

> $ 명령어 > 파일
>
> 명령어의 표준출력을 모니터 대신에 파일에 저장한다.

- 예

```
$ who > names.txt
$ cat names.txt

$ ls / > list.txt
$ cat list.txt
```



출력 재지정 이용: 간단한 파일 만들기

- 사용법

```
$ cat > 파일
```

표준입력 내용을 모두 파일에 저장한다. 파일이 없으면 새로 만든다.

- 예

```
$ cat > list1.txt
Hi !
This is the first list.
^D

$ cat > list2.txt
Hello !
This is the second list.
^D
```

두 개의 파일을 붙여서 새로운 파일 만들기

- 사용법

```
$ cat 파일1 파일2 > 파일3
```

파일1과 파일2의 내용을 붙여서 새로운 파일3을 만들어 준다.

- 예

```
$ cat list1.txt list2.txt > list3.txt
$ cat list3.txt
Hi !
This is the first list.
Hello !
This is the second list.
```

출력 추가

- 사용법

```
$ 명령어 >> 파일
```

명령어의 표준출력을 모니터 대신에 파일에 추가한다.

- 예

```
$ date >> list1.txt
$ cat list1.txt
Hi !
This is the first list.
Fri Sep 2 18:45:26 KST 2016
```

입력 재지정(input redirection)

- 사용법

  $ 명령어 < 파일

  명령어의 표준입력을 키보드 대신에 파일에서 받는다.

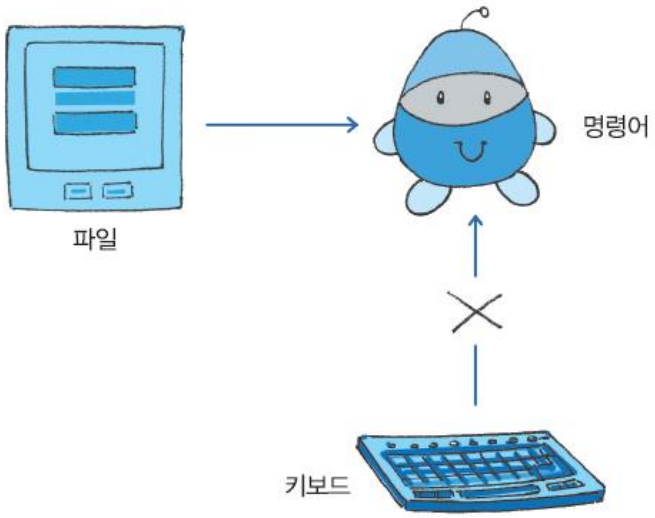
- 예

```
$ wc < list1.txt
3 13 58 list1.txt
```

- 참고

```
$ wc
...
^D

$ wc list1.txt
```



문서 내 입력(here document)

- 사용법

```
$ 명령어 << 단어
...
단어
```

명령어의 표준입력을 키보드 대신에 단어와 단어 사이의 입력 내용으로 받는다.

- 예

```
$ wc << END
hello !
word count
END
2    4      20
```

오류 재지정

- 사용법

> $ 명령어 2> 파일
>
> 명령어의 표준오류를 모니터 대신에 파일에 저장한다.

- 명령어의 실행결과
  - 표준출력(standard output): 정상적인 실행의 출력
  - 표준오류(standard error): 오류 메시지 출력

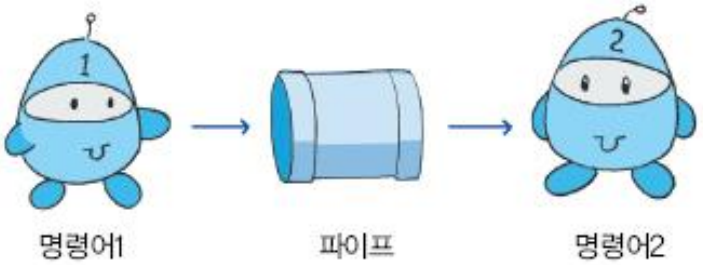
- 사용법

```
$ ls -l /bin/usr 2> err.txt
$ cat err.txt
ls: cannot access /bin/usr: No such file or directory
```

파이프

- 로그인 된 사용자들을 정렬해서 보여주기

```
$ who > names.txt
$ sort < names.txt
```



- 사용법

> $ 명령어1 | 명령어2
>
> 명령어1의 표준출력이 파이프를 통해 명령어2의 표준입력이 된다.

- 예

```
$ who | sort
agape pts/5 2월 20일 13:23 (203.252.201.55)
chang pts/3 2월 20일 13:28 (221.139.179.42)
hong pts/4 2월 20일 13:35 (203.252.201.51)
```

파이프 사용 예

- 예: 로그인 된 사용자 이름 정렬

```
$ who | sort
agape pts/5 2월 20일 13:23 (203.252.201.55)
chang pts/3 2월 20일 13:28 (221.139.179.42)
hong pts/4 2월 20일 13:35 (203.252.201.51)
```

- 예: 로그인 된 사용자 수 출력

```
$ who | wc -l
3
```

- 예: 특정 디렉터리 내의 파일의 개수 출력

```
$ ls 디렉터리 | wc -w
```

입출력 재지정 관련 명령어 요약

명령어 사용법	의미
명령어 > 파일	명령어의 표준출력을 모니터 대신에 파일에 추가한다.
명령어 >> 파일	명령어의 표준출력을 모니터 대신에 파일에 추가한다.
명령어 < 파일	명령어의 표준입력을 키보드 대신에 파일에서 받는다.
명령어 << 단어 . . . 단어	표준입력을 키보드 대신에 단어와 단어 사이의 입력 내용으로 받는다.
명령어 2> 파일	명령어의 표준오류를 모니터 대신에 파일에 저장한다.
명령어1 | 명령어2	명령어1의 표준출력이 파이프를 통해 명령어2의 표준입력이 된다.
cat 파일1 파일2 > 파일3	파일1과 파일2의 내용을 붙여서 새로운 파일3을 만들어준다.

5.5 여러 개 명령어 사용하기

명령어 열(command sequence)

- **명령어 열**
  - 나열된 명령어들을 순차적으로 실행한다.
- 사용법

> $ 명령어1; … ; 명령어n
>
> 나열된 명령어들을 순차적으로 실행한다.

- 예

```
$ date; pwd; ls
Fri Sep 2 18:08:25 KST 2016
/home/chang/linux/test
list1.txt list2.txt list3.txt
```

명령어 그룹(command group)

- **명령어 그룹**
  - 나열된 명령어들을 하나의 그룹으로 묶어 순차적으로 실행한다.
- 사용법

> $ (명령어1; … ; 명령어n)
>
> 나열된 명령어들을 하나의 그룹으로 묶어 순차적으로 실행한다.

- 예

```
$ date; pwd; ls > out1.txt
Fri Sep 2 18:08:25 KST 2016
/home/chang/linux/test
$ (date; pwd; ls) > out2.txt
$ cat out2.txt
Fri Sep 2 18:08:25 KST 2016
/home/chang/linux/test
...
```

조건 명령어 열(conditional command sequence)

- **조건 명령어 열**
  - 첫 번째 명령어 실행 결과에 따라 다음 명령어 실행을 결정할 수 있다.
- 사용법

> $ 명령어1 && 명령어2
>
> 명령어1이 성공적으로 실행되면 명령어2가 실행되고, 그렇지 않으면 명령어2가 실행되지 않는다.

- 예

```
$ gcc myprog.c && a.out
```

조건 명령어 열

- 사용법

> $ 명령어1 || 명령어2
>
> 명령어1이 실패하면 명령어2가 실행되고, 그렇지 않으면 명령어2가 실행되지 않는다.

- 예

```
$ gcc myprog.c || echo 컴파일 실패
```

여러 개 명령어 사용: 요약

명령어 사용법	의미
명령어1; … ; 명령어n	나열된 명령어들을 순차적으로 실행한다.
(명령어1; … ; 명령어n)	나열된 명령어들을 하나의 그룹으로 묶어 순차적으로 실행한다.
명령어1 && 명령어2	명령어1이 성공적으로 실행되면 명령어2가 실행되고, 그렇지 않으면 명령어2가 실행되지 않는다.
명령어1 || 명령어2	명령어1이 실패하면 명령어2가 실행되고, 그렇지 않으면 명령어2가 실행되지 않는다.

5.6 파일 이름 대치와 명령어 대치

파일 이름 대치

- 대표문자를 이용한 파일 이름 대치
  - 대표문자를 이용하여 한 번에 여러 파일들을 나타냄
  - 명령어 실행 전에 대표문자가 나타내는 파일 이름들로 먼저 대치하고 실행

대표문자	의미
*	빈 스트링을 포함하여 임의의 스트링을 나타냄
?	임의의 한 문자를 나타냄
[..]	대괄호 사이의 문자 중 하나를 나타내며 부분범위 사용 가능함.

```
$ gcc *.c
$ gcc a.c b.c test.c

$ ls *.txt
$ ls [ac]*
```

명령어 대치(command substitution)

- 명령어를 실행할 때 다른 명령어의 실행 결과를 이용
  - `명령어` 부분은 그 명령어의 실행 결과로 대치된 후에 실행
- 예

```
$ echo 현재 시간은 `date`

$ echo 현재 디렉터리 내의 파일의 개수 : `ls | wc -w`
현재 디렉터리 내의 파일의 개수 : 32
```

따옴표 사용

- 따옴표를 이용하여 대치 기능을 제한

```
$ echo 3 * 4 = 12
3 cat.csh count.csh grade.csh invite.csh menu.csh test.sh = 12
$ echo "3 * 4 = 12"
3 * 4 = 12
$ echo '3 * 4 = 12'
3 * 4 = 12

$ name=나가수
$ echo '내 이름은 $name 현재 시간은 `date`'
내 이름은 $name 현재 시간은 `date`
$ echo "내 이름은 $name 현재 시간은 `date`"
내 이름은 나가수 현재 시간은 2016. 11. 11. (금) 10:27:48 KST
```

따옴표 사용

- 정리
  1. 작은따옴표(')는 파일이름 대치, 변수 대치, 명령어 대치를 모두 제한한다.
  2. 큰따옴표(")는 파일이름 대치만 제한한다.
  3. 따옴표가 중첩되면 밖에 따옴표가 효력을 갖는다.

핵심 개념

- 쉘은 사용자와 운영체제 사이에 창구 역할을 하는 소프트웨어로 사용자로부터 명령어를 입력받아 이를 처리하는 명령어 처리기 역할을 한다.
- 출력 재지정은 표준출력 내용을 파일에 저장하고 입력 재지정은 표준입력을 파일에서 받는다.
- 파이프를 이용하면 한 명령어의 표준출력을 다른 명령어의 표준입력으로 바로 받을 수 있다.