4. Process and Redirection

Hyunchan, Park

http://oslab.jbnu.ac.kr

Division of Computer Science and Engineering

Jeonbuk National University

학습 내용

- 프로세스 관리와 관련 명령어 실습
  - 프로세스 기초 및 상태 보기
  - 작업 제어, 전면 및 후면 작업
- 입출력 재지정과 관련 명령어 실습
- 다양한 명령어 사용 방법과 실습

개인 과제 4: 실습 및 C언어 복습

- 실습 과제
  - 실습 내용에서 다루는 명령어를 모두 입력하고, 그 결과를 확인할 것
    - 동영상에서 수행한 내용
  - 제출 방법
    - Old LMS, 개인 과제 4
    - Xshell 로그 파일 1개 제출
    - 파일 명: 학번.txt
- C언어 복습 과제
  - JOTA: hw4-1, hw4-2 문제 수행
- 제출 기한
  - 10/5 (월) 23:59 (지각 감점: 5%p / 12H, 1주 이후 제출 불가)

Process: Basic

프로세스(process)

- 실행중인 프로그램을 **프로세스**(process)라고 부른다.
- 각 프로세스는 유일한 프로세스 번호 PID를 갖는다.
- 각 프로세스는 부모 프로세스에 의해 생성된다.

프로세스의 사용자 ID

- 프로세스는 프로세스 ID 외에
- 프로세스의 사용자 ID (UID)와 그룹 ID (GID)를 갖는다.
  - 해당 프로세스를 실행시킨 사용자의 ID와 사용자의 그룹 ID
  - 해당 사용자 및 그룹의 권한을 상속 받아,
  - 프로세스가 수행할 수 있는 연산을 결정하는 데 사용된다.

- id 명령어

```
$ id [사용자명]
사용자의 실제 ID와 유효 사용자 ID, 그룹 ID 등을 보여준다.
```

  - $ id
  - uid=1000(chang) gid=1002(cs) groups=1002(cs)
    context=system_u:unconfined_r:unconfined_t:s0

프로세스의 사용자 ID

- 프로세스의 실제 사용자 ID(real user ID)
  - 그 프로세스를 실행시킨 사용자의 ID로 설정된다.
  - 예: chang 사용자 ID로 로그인하여 어떤 프로그램을 실행시키면
  - 그 프로세스의 실제 사용자 ID는 chang이 된다.

- 프로세스의 유효 사용자 ID(effective user ID)
  - 현재 유효한 사용자 ID
  - 보통 유효 사용자 ID와 실제 사용자 ID는 같다.
  - 새로 파일을 만들 때나 파일의 접근권한을 검사할 때 주로 사용됨
  - 특별한 실행파일을 실행할 때 유효 사용자 ID는 달라진다.

set-user-id 실행파일

- set-user-id(set user ID upon execution) 실행권한
  - set-user-id가 설정된 실행파일을 실행하면
  - 이 프로세스의 유효 사용자 ID는 그 실행파일의 소유자로 바뀜.
  - 이 프로세스는 실행되는 동안 그 파일의 소유자 권한을 갖게 됨.

- 예
  - $ ls –l /usr/bin/passwd
  - -rwsr-xr-x. 1 root root 27000 2010-08-22 12:00 /usr/bin/passwd

  - set-user-id 실행권한이 설정된 실행파일이며 소유자는 root
  - 일반 사용자가 이 파일을 실행하게 되면 이 프로세스의 유효 사용자 ID는 root가 됨.
  - /etc/passwd처럼 root만 수정할 수 있는 파일의 접근 및 수정 가능

set-group-id 실행파일

- set-group-id(set group ID upon execution) 실행권한
  - 실행되는 동안에 그 파일 소유자의 그룹을 프로세스의 유효 그룹 ID으로 갖게 된다.
  - set-group-id 실행권한은 8진수 모드로는 2000으로 표현된다.

- set-group-id 실행파일 예

  $ ls -l /usr/bin/wall

  -r-xr-sr-x. 1 root tty 15344 6월 10 2014 /usr/bin/wall

set-user-id/set-group-id 설정

- set-user-id 실행권한 설정

  $ chmod 4755 파일 혹은 $ chmod u+s 파일

- set-group-id 실행권한 설정

  $ chmod 2755 파일 혹은 $ chmod g+s 파일

프로세스 상태 보기: ps(process staus)

- 사용법

$ ps [-옵션]

현재 시스템 내에 존재하는 프로세스들의 실행 상태를 요약해서 출력한다.

-A Select all processes. Identical to -e

-a Select all processes except both session leaders (see getsid(2)) and processes not associated with a terminal. (shell, terminal 등 제외)

-u [userlist] Select by effective user ID (EUID) or name. “x” means everyone.

-e Select all processes. Identical to -A.

-f Do full-format listing.

프로세스 상태 보기: ps(process staus)

- 자주 사용하는 형태
  - $ ps
    - 현재 사용자가 생성한 프로세스 목록을 간단히 출력
  - $ ps -aux
    - 모든 프로세스 출력. 메모리, CPU 사용량, 프로세스 상태
  - $ ps -ef
    - 모든 프로세스 출력. 자식 프로세스 개수, 부모 프로세스 ID
    - 부모 자식 간의 관계를 파악하기 더 편리함
  - $ ps -aux --sort [+|-]key
    - Key 로 지정된 필드 순으로 정렬하여 출력. Default: PID
    - +|-: 오름차순 혹은 내림차순. Default: 오름차순
    - 예제)
      - $ ps -aux --sort user : 사용자 순으로 정렬.
      - $ ps -aux --sort -rss : 실제 메모리 사용량이 많은 순으로 정렬

ps -aux

```
$ ps -aux
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
root 1 0.0 0.0 2064 652 ? Ss 2011 0:27 init [5]
root 2 0.0 0.0 0 0 ? S< 2011 0:01 [migration/0]
root 3 0.0 0.0 0 0 ? SN 2011 0:00 [ksoftirqd/0]
root 4 0.0 0.0 0 0 ? S< 2011 0:00 [watchdog/0]

...

root 8692 0.0 0.1 9980 2772 ? Ss 11:12 0:00 sshd: chang [pr
chang 8694 0.0 0.0 9980 1564 ? R 11:12 0:00 sshd: chang@pts
chang 8695 0.0 0.0 5252 1728 pts/3 Ss 11:12 0:00 bash
chang 8976 0.0 0.0 4252  940 pts/3 R+ 11:24 0:00 ps aux
```

ps 출력 정보

항목	의미
UID	프로세스를 실행시킨 사용자 ID
PID	프로세스 번호
PPID	부모 프로세스 번호
C	프로세스의 우선순의
STIME	프로세스의 시작 시간
TTY	명령어가 시작된 터미널
TIME	프로세스에 사용된 CPU 시간
CMD	실행되고 있는 명령어(프로그램) 이름

특정 프로세스 리스트: pgrep

- 특정 프로세스만 리스트

- 사용법

```
$ pgrep [옵션] [패턴]
패턴에 해당하는 프로세스들만을 리스트 한다.
 -l : PID와 함께 프로세스의 이름을 출력한다.
 -f : 명령어의 경로도 출력한다.
 -n : 패턴과 일치하는 프로세스들 중에서 가장 최근 프로세스만을 출력한다
 -x : 패턴과 정확하게 일치되는 프로세스만 출력한다.
```

특정 프로세스 리스트: pgrep

- 예

  $ pgrep sshd

  1720

  1723

  5032

- -l 옵션: 프로세스 번호와 프로세스 이름을 함께 출력

  $ pgrep -l sshd

  1720 sshd

  1723 sshd

  5032 sshd

- -n 옵션: 가장 최근 프로세스만 출력한다.

  $ pgrep -ln sshd

  5032 sshd

프로세스 트리 출력

- 사용법

```
$ pstree
```

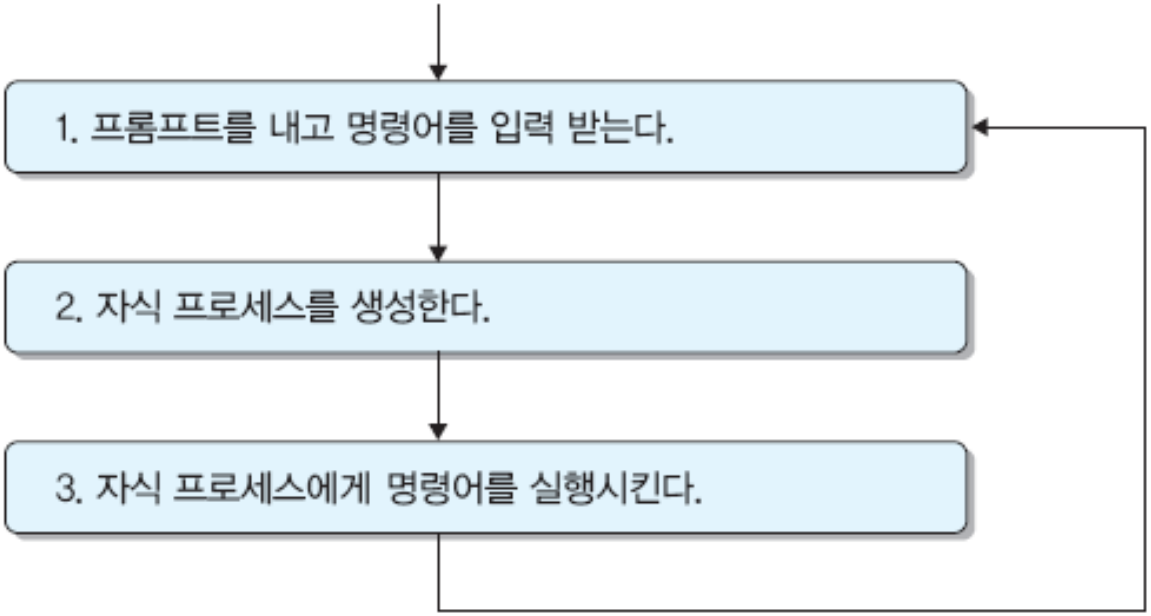
실행중인 프로세스들의 부모, 자식 관계를 트리 형태로 출력한다.

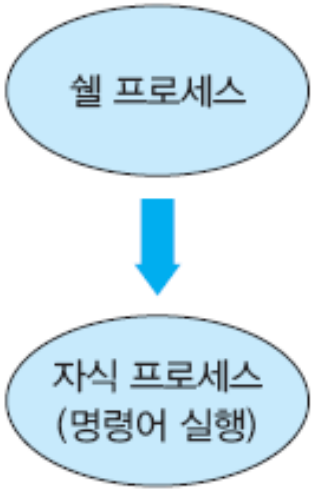
```
root@unix:/home/ubuntu# pstree
systemd─┬─accounts-daemon───2*[{accounts-daemon}]
        ├─2*[agetty]
        ├─atd
        ├─cron
        ├─dbus-daemon
        ├─irqbalance───{irqbalance}
        ├─multipathd───6*[{multipathd}]
        ├─networkd-dispat
        ├─polkitd───2*[{polkitd}]
        ├─rsyslogd───3*[{rsyslogd}]
        ├─snapd───16*[{snapd}]
        ├─sshd─┬─3*[sshd───sshd───bash]
        │      └─sshd───sshd───bash───sudo───bash───pstree
        ├─4*[systemd───(sd-pam)]
        ├─systemd-journal
        ├─systemd-logind
        ├─systemd-network
        ├─systemd-resolve
        ├─systemd-timesyn───{systemd-timesyn}
        ├─systemd-udevd
        └─unattended-upgr───{unattended-upgr}
root@unix:/home/ubuntu#
```

Process: Control

쉘과 수행된 프로세스 간의 관계

- 쉘에서 수행되는 프로세스는 쉘 프로세스의 자식 프로세스





쉘에 메시지 출력

- 사용법

```
$ echo 메시지
메시지를 화면에 출력함
```

- 예

```
$ echo DONE
```

```
ubuntu@unix:~$ echo DONE
DONE
ubuntu@unix:~$
```

쉘 재우기

- 사용법

```
$ sleep 초
명시된 시간만큼 프로세스 실행을 중지시킨다.
```

- 예

```
$ echo Start; sleep 5; echo End
```

```
ubuntu@unix:~$ echo Start; sleep 5; echo Done
Start
Done
ubuntu@unix:~$
```

강제 종료

- **강제 종료 Ctrl-C**

  $ 명령어

  ^C

- **실행 중단 Ctrl-Z**

  $ 명령어

  ^Z

  [1]+ Stopped 명령어

```
ubuntu@unix:~$ echo Start; sleep 5; echo Done
Start
^C
ubuntu@unix:~$ echo Start; sleep 5; echo Done
Start
^Z
[3]+  Stopped                 sleep 5
Done
ubuntu@unix:~$
```

전면 처리 vs. 후면처리

- 전면 처리 (일반적인 사용법)
  - 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.
  - 사용법: 명령어 뒤에 “&” 를 붙여 실행
  - $ 명령어 **&**
  - 예) $ (echo Start; sleep 5; echo End) &

    [1] 8320 <- 후면 작업의 Job 번호와 PID 가 출력됨
    - 참고: () 를 통해 3개 명령어 전체가 background 로 수행됨

실행 예

- 후면 처리의 출력
  - 입력 대기 상태인 shell process 가 foreground 에서 수행 중 일 때,
  - Background 에서 출력되는 메시지는 바로 shell 에 표시됨

```
ubuntu@unix:/$ (echo Start; sleep 5; echo End) &
[1] 244368
ubuntu@unix:/$ Start

ubuntu@unix:/$ End

[1]+  Done                    ( echo Start; sleep 5; echo End )
ubuntu@unix:/$
```

후면 작업 확인

- 사용법

```
$ jobs [%작업번호]
```

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

```
ubuntu@unix:/$ vi
ubuntu@unix:/$ vi &
[1] 244536
ubuntu@unix:/$ jobs
[1]+  Stopped                 vi
ubuntu@unix:/$ (echo Start; sleep 5; echo End) &
[2] 244546
Start
ubuntu@unix:/$ jobs
[1]+  Stopped                 vi
[2]-  Running                 ( echo Start; sleep 5; echo End ) &
ubuntu@unix:/$ End
```

후면 작업을 전면 작업으로 전환

- 사용법

```
$ fg %작업번호
```

작업번호에 해당하는 후면 작업을 전면 작업으로 전환시킨다.

- 예

```
ubuntu@unix:/$ (echo Start; sleep 10; echo End) &
[1] 244568
Start
ubuntu@unix:/$ fg %1
( echo Start; sleep 10; echo End )
End
ubuntu@unix:/$
```

전면 작업의 후면 전환: bg(background)

- 사용법
  - Ctrl-Z 키를 눌러 전면 실행중인 작업을 먼저 중지시킨 후
  - bg 명령어 사용하여 후면 작업으로 전환

```
$ bg %작업번호
작업번호에 해당하는 중지된 작업을 후면 작업으로 전환하여 실행한다.
```

- 예

```
ubuntu@unix:/$ (echo Start; sleep 10; echo End)
Start
^Z
[1]+  Stopped                 ( echo Start; sleep 10; echo End )
ubuntu@unix:/$ jobs
[1]+  Stopped                 ( echo Start; sleep 10; echo End )
ubuntu@unix:/$ bg %1
[1]+ ( echo Start; sleep 10; echo End ) &
ubuntu@unix:/$ End

[1]+  Done                    ( echo Start; sleep 10; echo End )
```

프로세스 끝내기: kill

- 프로세스 강제 종료

```
$ kill 프로세스번호
$ kill %작업번호
```

프로세스 번호(혹은 작업 번호)에 해당하는 프로세스를 강제로 종료시킨다.

- 예

```
$ (sleep 100; echo done) &
[1] 8320
  $ kill 8320 혹은 $ kill %1
  [1] Terminated ( sleep 100; echo done )
```

프로세스 기다리기: wait

- 사용법

```
$ wait [프로세스번호]
프로세스 번호로 지정한 자식 프로세스가 종료될 때까지 기다린다.
지정하지 않으면 모든 자식 프로세스가 끝나기를 기다린다.
```

- 예

```
$ (sleep 10; echo 1번 끝) &
[1] 1231
$ echo 2번 끝; wait 1231; echo 3번 끝
2번 끝
1번 끝
3번 끝
```

프로세스 기다리기: wait

- 예

```
$ (sleep 10; echo 1번 끝) &
$ (sleep 10; echo 2번 끝) &
$ echo 3번 끝; wait; echo 4번 끝
3번 끝
1번 끝
2번 끝
4번 끝
```

Redirection

출력 재지정(output redirection)

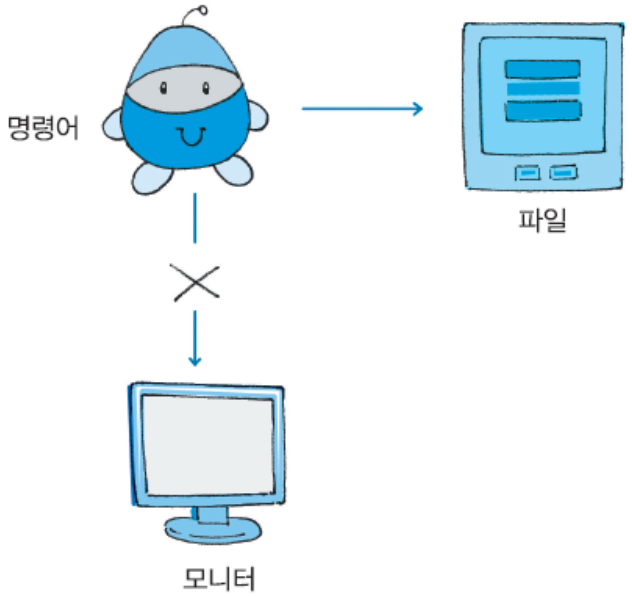
- 사용법

> $ 명령어 > 파일
>
> 명령어의 표준출력(stdout: standard output)을 모니터 대신에 파일에 저장한다.
>
> 파일이 없으면 새로 만들고, 있다면 기존 내용을 덮어쓴다. (overwrite)

- 예

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

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



* who : show who is logged on the system

* last : show a listing of last logged in users

출력 재지정 이용: 간단한 파일 만들기

- 사용법

> $ cat > 파일
>
> 표준입력 (stdin: standard input) 내용을 모두 파일에 저장한다.

- 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 추가(append)한다.

- 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.

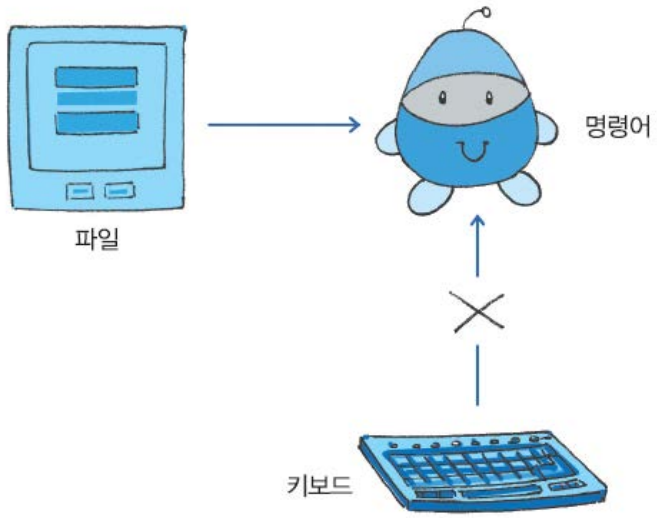
- 예

```
$ 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.

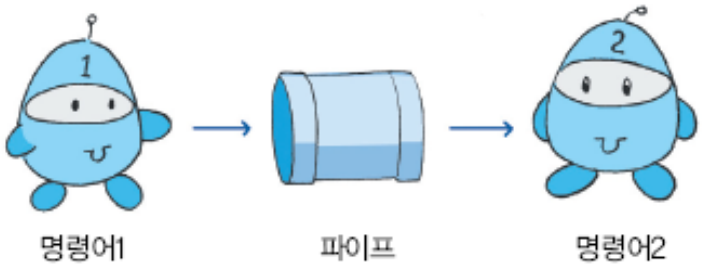
- 명령어의 실행결과
  - 표준출력(standard output): 정상적인 실행의 출력
  - 표준오류(standard error): 오류 메시지 출력

- 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.

Commanding

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.

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
  - ` : back quote, backtick (역 따옴표)

- 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

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.