# 제2장 리눅스 사용

# 2.3 파일 속성

# 파일 속성(file attribute)

- 파일의 이름, 타입, 크기, 소유자, 사용권한, 수정 시간

  $ ls -sl cs1.txt

  6 -rw-r--r-- 1 chang faculty 2088 4월 16일 13:37 cs1.txt

| 파일 속성 | 의미 |
|---|---|
| 블록 수 | 파일의 블록 수 |
| 파일 타입 | 일반 파일(-), 디렉터리(d), 링크(l), 파이프(p), 소켓(s), 디바이스(b 혹은 c) 등의 파일 종류를 나타낸다. |
| 사용권한 | 소유자, 그룹, 기타 사용자의 파일에 대한 읽기/쓰기/실행 권한 |
| 소유자 및 그룹 | 파일의 소유자 및 소유자가 속한 그룹 |
| 크기 | 파일을 구성하는 블록 수 → 파일을 구성하는 블록의 실제 크기 (파일의 크기) |
| 수정 시간 | 파일을 최후로 생성 혹은 수정한 시간 |

# 파일 종류

- 리눅스에서 지원하는 파일 종류

| 파일 종류 | 표시 | 설명 |
|---|---|---|
| 일반 파일 | - | 데이터를 갖고 있는 텍스트 파일 또는 이진 파일 |
| 디렉터리 파일 | d | 디렉터리 내의 파일들의 이름들과 파일 정보를 관리하는 파일 |
| 문자 장치 파일 | c | 문자 단위로 데이터를 전송하는 장치를 나타내는 파일 |
| 블록 장치 파일 | b | 블록 단위로 데이터를 전송하는 장치를 나타내는 파일 |
| FIFO 파일 | p | 프로세스 간 통신에 사용되는 이름 있는 파이프 |
| 소켓 | s | 네트워크를 통한 프로세스 간 통신에 사용되는 파일 |
| 심볼릭 링크 | l | 다른 파일을 가리키는 포인터와 같은 역할을 하는 파일 |

# 파일 종류

- 사용법

```
$ file 파일
```

파일의 종류에 대한 자세한 정보를 출력한다.

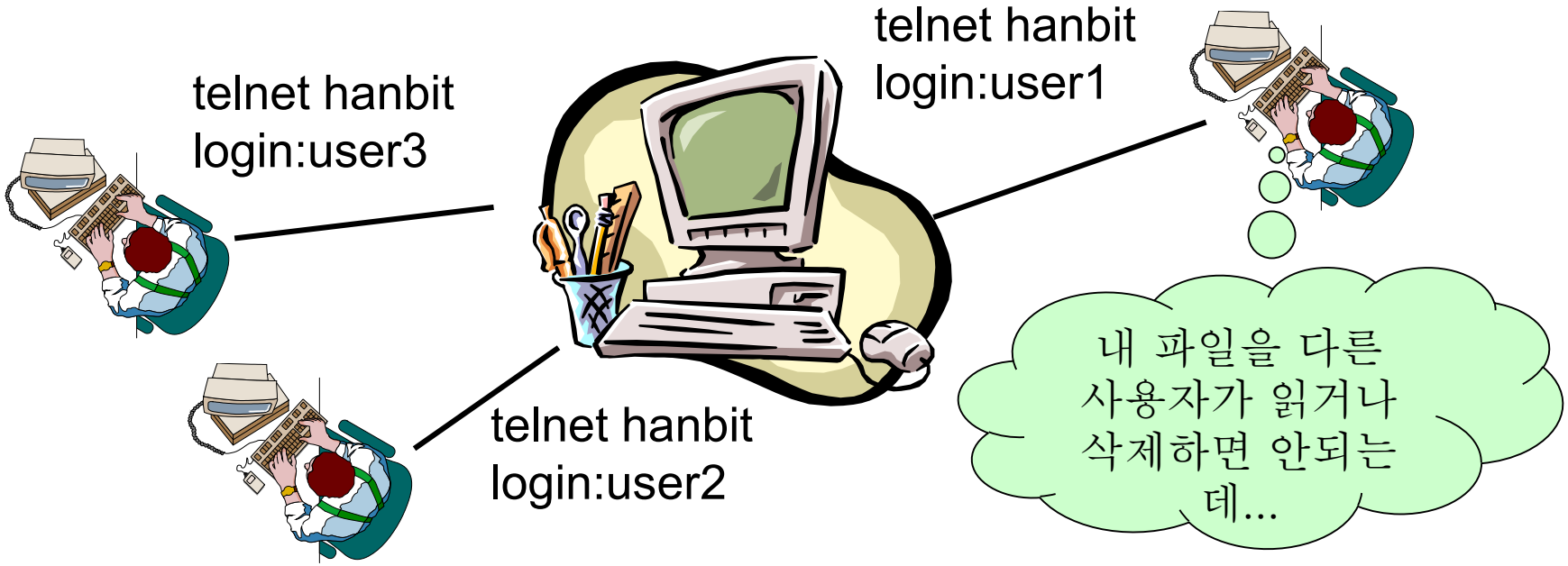
- 예

파일이 비어있으면 empty

```
$ file cs1.txt
cs1.txt: ASCII text
$ file a.out
a.out: ELF 32-bit LSB executable, ...
```

# 사용권한(permission mode)

- 다중 사용자 시스템



# 사용권한(permission mode)

- 읽기(r), 쓰기(w), 실행(x) 권한

| 권한 | 파일 | 디렉터리 |
|---|---|---|
| r | 파일에 대한 읽기 권한 | 디렉터리 내에 있는 파일명을 읽을 수 있는 권한 |
| w | 파일에 대한 쓰기 권한 | 디렉터리 내에 파일을 생성하거나 삭제할 수 있는 권한 |
| x | 파일에 대한 실행 권한 | 디렉터리 내로 탐색을 위해 이동할 수 있는 권한 |

- 파일의 사용권한은 소유자(owner)/그룹(group)/기타(others)로 구분하여 관리한다.

- 예

  소유자 그룹 기타

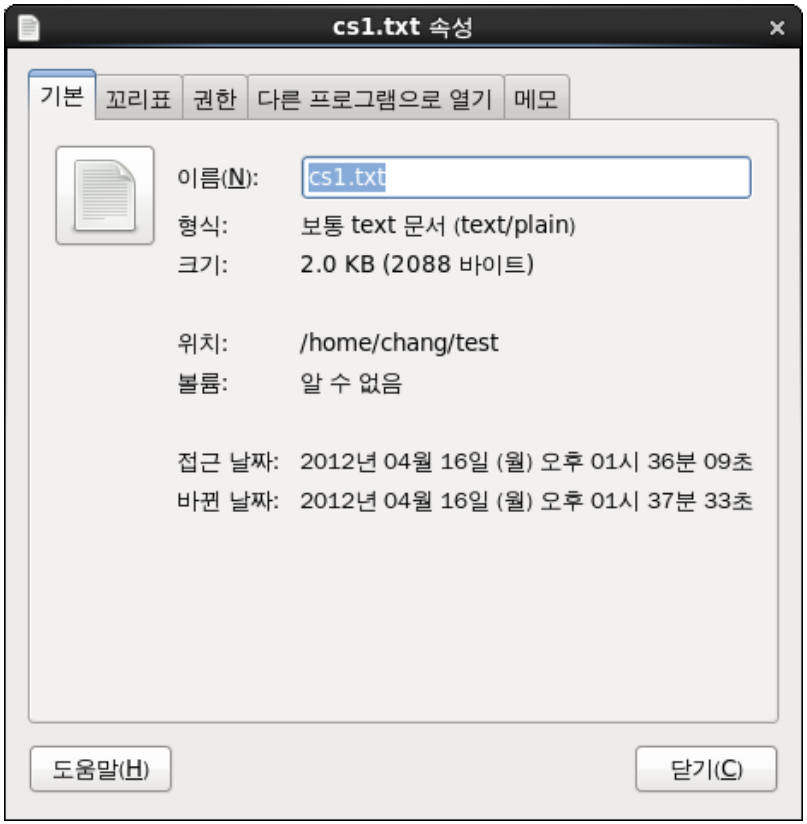
  rw- r-- r--

# 접근권한의 예

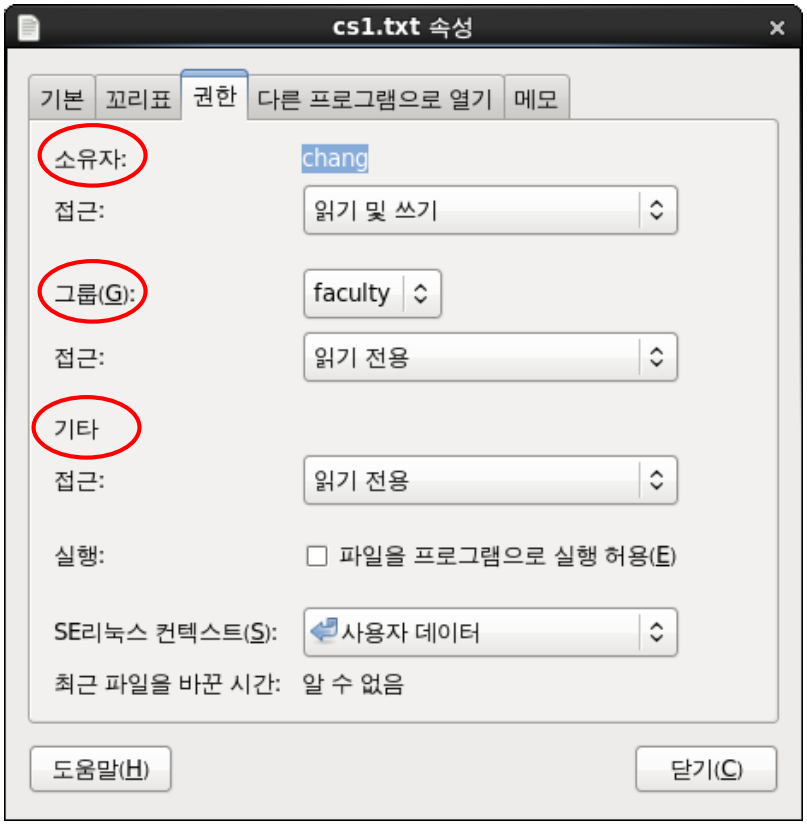
| 접근권한 | 의미 |
|---|---|
| rwxrwxrwx | 소유자, 그룹, 기타 사용자 모두 읽기,쓰기,실행 가능 |
| rwxr-xr-x | 소유자만 읽기,쓰기,실행 가능,<br>그룹, 기타 사용자는 읽기,실행 가능 |
| rw-rw-r-- | 소유자와 그룹만 읽기,쓰기 가능,<br>기타 사용자는 읽기만 가능 |
| rw-r--r-- | 소유자만 읽기,쓰기 가능,<br>그룹과 기타 사용자는 읽기만 가능 |
| rw-r----- | 소유자만 읽기,쓰기 가능<br>그룹은 읽기만 가능 |
| rwx------ | 소유자만 읽기,쓰기,실행 가능 |

# X 윈도우의 GNOME 데스크톱에서 속성 확인

## 기본 속성




## 사용권한



# chmod(change mode)

- 파일 혹은 디렉토리의 사용권한을 변경하는 명령어

  $ chmod [-R] 사용권한 파일

  -R 옵션은 디렉토리 내의 모든 파일 및 하위 디렉토리에 대해서도 이 명령어를 적용함을 의미

- 사용권한 rw- rw- r--
- 2진수: 110 110 100
- 8진수: 6 6 4
- $ chmod 664 cs1.txt

- [u|g|o|a]+[+|-|=][r|w|x]+ (1개 이상)
- u(user), g(group), o(other), a(all)
- 연산자: +(추가), -(제거), =(지정)
- 권한: r(읽기), w(쓰기), x(실행)
- $ chmod g+w cs1.txt

기존 모든 권한 제거
지정된 권한만 남음

# 접근권한 변경: chmod(change mode)

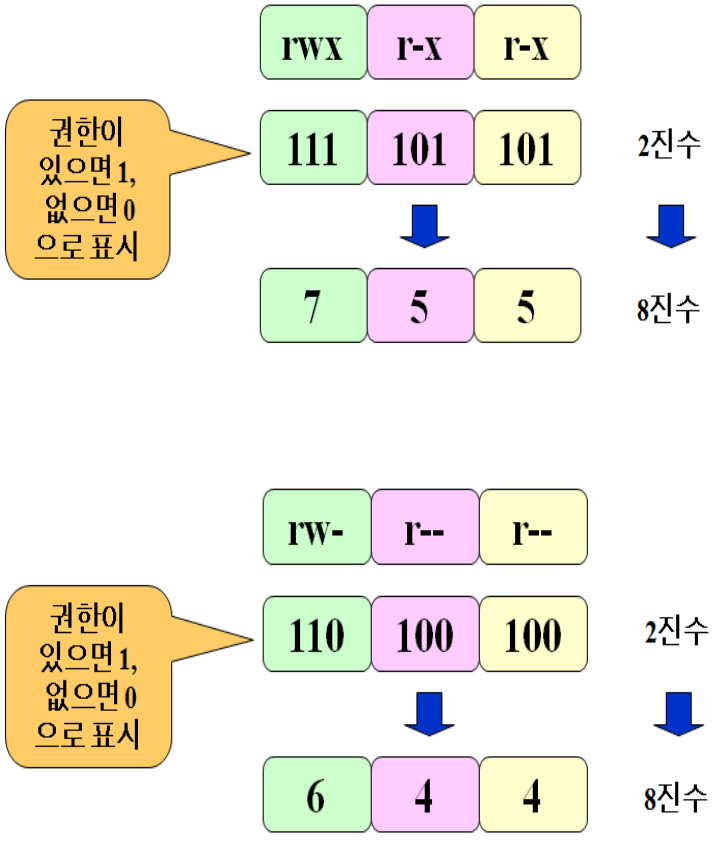
- 사용법

```
$ chmod [-R] 접근권한 파일 혹은 디렉터리
```

파일 혹은 디렉터리의 접근권한을 변경한다. -R 옵션을 사용하면 지정된 디렉터리 아래의 모든 파일과 하위 디렉터리에 대해서도 접근권한을 변경한다.

# 접근권한 표현: 8진수

- 접근권한 8진수 변환




- 사용 예

```
$ chmod 644 cs1.txt
$ ls -l cs1.txt
-rw-r--r-- 1 chang … cs1.txt
```

| 접근권한 | 8진수 |
|---|---|
| rwxrwxrwx | 777 |
| rwxr-xr-x | 755 |
| rw-rw-r-- | 664 |
| rw-r--r-- | 644 |
| rw-r----- | 640 |
| rwx------ | 700 |

# 접근권한 표현: **기호**

- 기호를 이용한 접근권한 변경

| 사용자범위 | 연산자 | 권한 |
|---|---|---|
| [u\|g\|o\|a]+ | [+\|-\|=] | [r\|w\|x]+ |

| 구분 | 기호와 의미 |
|---|---|
| 사용자 범위 | u(user:소유자), g(group:그룹), o(others:기타 사용자), a(all:모든 사용자) |
| 연산자 | +(권한 추가), -(권한 제거), =(권한 설정) |
| 권한 | r(읽기 권한), w(쓰기 권한), x(실행 권한) |

# 기호를 이용한 접근권한 변경

- 예

```
$ chmod g-w cs1.txt
$ ls -l cs1.txt
-rw-r--r-- 1 chang cs 2088 4월 16일 13:37 cs1.txt

$ chmod o-r cs1.txt
$ ls -l cs1.txt
-rw-r----- 1 chang cs 2088 4월 16일 13:37 cs1.txt

$ chmod g+w,o+rw cs1.txt
$ ls -l cs1.txt
-rw-rw-rw- 1 chang cs 2088 4월 16일 13:37 cs1.txt
```

# umask (기본 권한 설정)

user mask (권한을 없앰)

- 기본 권한 설정
  - 새로운 파일이 만들어질 때 적용되는 기본 권한
  - 마스크 값을 지정하지 않으면 현재의 마스크 값을 보여줌
  - 사용권한에서 허용하지 않을 값을 지정

**telnet hanbit.co.kr**

```
$ umask
22          0 : 허용하지 않은게 없다(모두 허용한다)
$ umask 077
$ umask     모두 막는다
77
```

022를 의미
077을 의미

# umask (기본 권한 설정)

| 파 일 | 기본 접근 허가권 |
|---|---|
| 실행할 수 없는 일반 파일<br>(문서 편집기로 생성한 파일) | 666 rw-rw-rw- |
| 실행할 수 있는 일반 파일 | 777 rwxrwxrwx |
| 디렉토리 | 777 rwxrwxrwx |

1) 최대권한 rw-rw-rw- 666
2) 마스크값(022) -----w--w- 022
3) 뺄셈결과 rw-r—r-- 644

| 마스크 값 | 실행할 수 없는 일반 파일 | 실행할 수 있는 일반 파일 | 디렉토리 | 의 미 |
|---|---|---|---|---|
| ----w--w-<br>022 | rw-r--r--<br>644 | rwxr-xr-x<br>755 | rwxr-xr-x<br>755 | 소유자는 모두 할 수 있고 그 이외의 사용자는 쓰기 금지 |
| ---rwxrwx<br>077 | 666-077 =><br>600 (예외) | rwx------<br>700 | rwx------<br>700 | 소유자 이외는 파일 접근 금지 |

# chown(change owner)/chgrp(change group)

- chown 명령어
  파일이나 디렉토리의 소유자를 변경할 때 사용한다

  $ chown 사용자 파일
  $ chown [-R] 사용자 디렉토리

- chgrp 명령어
  파일의 그룹을 변경할 수 있다

  $ chgrp 그룹 파일
  $ chgrp [-R] 그룹 디렉토리

- 파일의 소유자 또한 슈퍼 유저만이 사용 가능

# 소유자 변경: chown(change owner)

- 사용법

```
$ chown 사용자 파일
$ chown [-R] 사용자 디렉터리
```

파일 혹은 디렉터리의 소유자를 지정된 사용자로 변경한다.

-R 옵션: 디렉터리 아래의 모든 파일과 하위 디렉터리에 대해서도 소유자를 변경한다.

- 예

```
$ chown hong cs1.txt
chown: changing ownership of 'cs1.txt': 명령을 허용하지 않음
$ su
암호:
$ chown hong cs1.txt
$ ls -l cs1.txt
-rw-r--r--. 1 hong cs 2088 10월 21 16:25 cs1.txt
```

# 그룹 변경: chgrp(change group)

- 사용법

```
$ chgrp 그룹 파일

$ chgrp [-R] 그룹 디렉터리
```

파일 혹은 디렉터리의 그룹을 지정된 그룹으로 변경한다. -R 옵션을 사용하면 지정된 디렉터리 아래의 모든 파일과 하위 디렉터리에 대해서도 그룹을 변경한다.

# 2.4 입출력 재지정 및 파이프

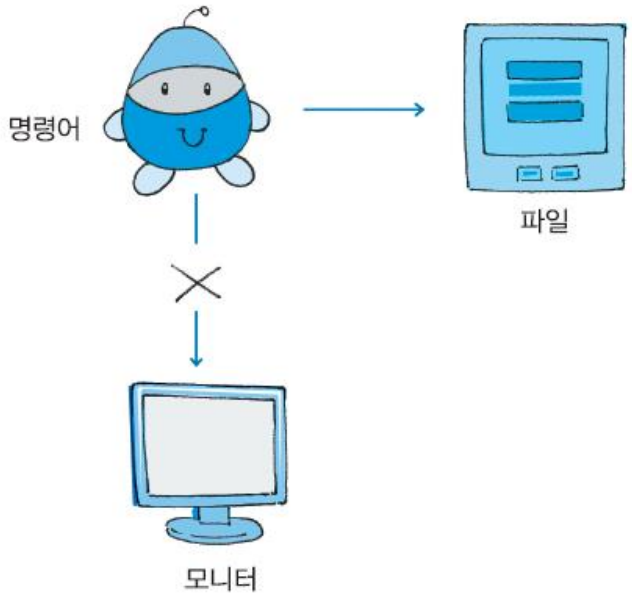
# 출력 재지정(output redirection)

- 명령어의 표준출력 내용을 모니터에 출력하는 대신에 파일에 저장

$ 명령어 > 파일

$ who > names.txt

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

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



# 출력 재지정 예

- $ cat > list1.txt
  Hi !
  This is the first list.
  ^D 끝
- $ cat > list2.txt
  Hello !
  This is the second list.
  ^D

cat a b
a,b 출력

- $ cat list1.txt list2.txt > list3.txt

결합

- $ cat list3.txt
  Hi !
  This is the first list.
  Hello !
  This is the second list.

# 출력 재지정 이용: 간단한 파일 만들기

- 사용법

원래 파일에 내용이 있다면
초기화 되고 적힘

```
$ 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

- 명령어의 표준출력을 모니터 대신에 기존 파일에 추가(append)

  $ 명령어 >> 파일

  원래 파일에 내용이 있다면
  초기화를 안시키고
  그 내용에 추가

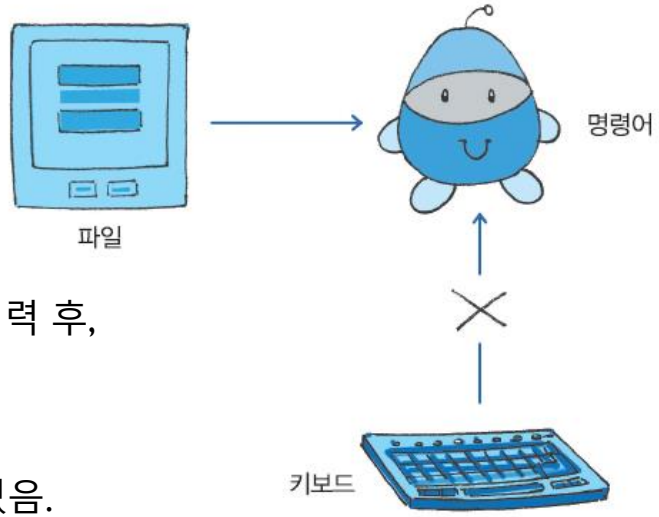
```
$ cat >> list1.txt
Bye !
This is the end of the first list.
^D

$ cat list1.txt
Hi !
This is the first list.
Bye !                                   // 여기서 부터 두 라인이 기존 파일 끝에 추가되었음
This is the end of the first list.
```

# 입력 재지정(input redirection)

- 명령어의 표준입력을 키보드 대신에 파일에서 받는다.

```
$ 명령어 < 파일
$ wc < list1.txt      // 사실상 wc list1.txt와 동일한 결과 출력
4 17 71 list1.txt
```

```
$ wc
...
^D
```




wc는 인자 없이 직접 입력 후,
$wc
…
ctrl-D
로 결과가 출력 될 수 있음.

# 문서 내 입력

- 명령어 실행 시, 입력을 문서 내에서 받을 수 있음
- 명령어의 표준입력을 문서내의 특정 단어가 나타날 때까지 수행
- 일반적으로 스크립트 내에서 입력을 받을 때 입력의 끝을 명시하기 위해 사용

```
$ 명령어 << 단어          // 단어가 나타날 때까지 내용을 입력으로 받음
    …
단어
```

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

<< : 특정 단어가 나타날 때까지 입력 받고 멈춤
문자열의 끝을 알리는 Null string 처럼,
스크립트에서 입력을 받을 때 입력의 끝을 지정하는 효과

# 오류 재지정

- 사용법

0: 입력
1: 출력
2: 오류

```
yu22312072@acslab-146:~$ cat asdf.txt 2
cat: asdf.txt: No such file or directory
cat: 2: No such file or directory
yu22312072@acslab-146:~$ cat list1.txt 2
Hi!
cat: 2: No such file or directory
```

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

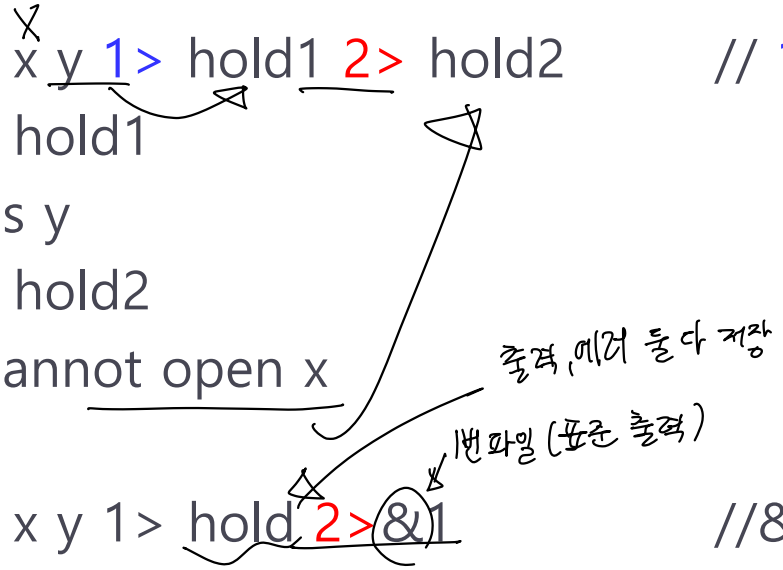
- 명령어의 실행결과
  - 표준출력(standard output): 정상적인 실행의 출력
  - 표준오류(standard error): 오류 메시지 출력

- 사용법

실행 중 오류 메세지를 → 표준 출력에 X, err.txt에 저장 O

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

# (고급) 표준 에러의 재지정




```
$ cat x y 1> hold1 2> hold2        // 1> 과 >은 동일
$ cat hold1
This is y
$ cat hold2
cat: cannot open x

$ cat x y 1> hold 2>&1             //& 띄어쓰기 없음 주의
```

표준 출력은 hold, file descriptor 2는 file descriptor 1의 사본
즉, 표준출력과 표준에러는 hold

Bash shell의 경우 사례이며, C shell은 사용법이 다름

어떤의미
언제 활용

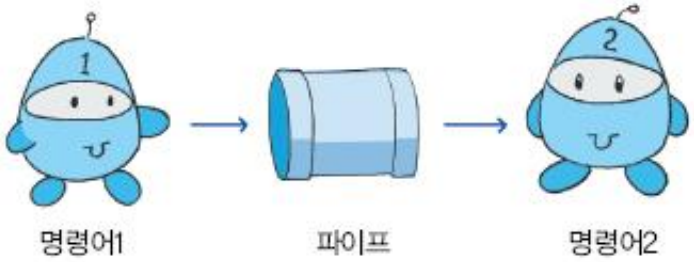
# 파이프

- 로그인 된 사용자들을 정렬해서 보여주기

  $ who > names.txt

  $ sort < names.txt

- $ 명령어1 | 명령어2
  - 명령어1의 표준출력을 명령어2의 표준입력으로 바로 받는다.




$ who | sort  // 위 두 명령어를 하나로 통합

$ who | wc -l 로그인한 수

# 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
```

- 응용 예: 특정 디렉터리 내의 파일의 개수 출력

```
$ ls 디렉터리 | wc -w
```

단어 개수

# 입출력 재지정 관련 명령어 요약

| 명령어 사용법 | 의미 |
|---|---|
| 명령어 > 파일 | 명령어의 표준출력을 모니터 대신에 파일에 신규출력한다. |
| 명령어 >> 파일 | 명령어의 표준출력을 모니터 대신에 파일에 추가한다. |
| 명령어 < 파일 | 명령어의 표준입력을 키보드 대신에 파일에서 받는다. |
| 명령어 << 단어(1)<br>. . .<br>단어(2) | 표준입력을 키보드 대신에 단어(1)와 단어(2) 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. |

# 2.5 후면 처리 및 프로세스

# 전면 처리 vs. 후면처리

- 전면 처리
  - 명령어를 입력하면 명령어가 전면에서 실행되며, 명령어 실행이 끝날 때까지 쉘이 기다려 준다.
  - 전면 처리를 강제 종료하기 위해서는 control-C 입력, 실행 중단을 위해서는 control-Z, 중단 명령어의 재실행은 fg
    - fg : 제일 최근 job을 전면에서 실행 재계

control-D
입력 종료
control-D 누르면 로그아웃

- 후면 처리
  - 명령어들을 후면에서 처리하고 전면에서는 다른 작업을 할 수 있으면 동시에 여러 작업을 수행할 수 있다.
  - 시간이 오래 걸리는 작업등의 실행에 유용
  - $ 명령어 &

# 후면 처리 예

(sleep 3;) (;있거나 없거나 똑같음)

따음표, 작은 따음표, 큰 따음표 결과 같음

- $ (sleep 100; echo "done") &
  [1] 8320

  괄호가 없으면? echo done만 후면 처리
  괄호가 있으면 모두 후면 처리
- $ find . -name test.c -print &
  [2] 8325

  fg %1 화면 앞으로 나오게 하는것
- $ jobs (후면 실행 작업 목록 표시)
  [1] - Running ( sleep 100; echo done )
  [2] + Running find . -name test.c –print // + : 가장 최근 작업 (교재 오류)
- $ fg %작업번호 (후면 실행 작업 중 하나를 선택하여 전면으로 실행)
  $ fg %1
  ( sleep 100; echo done )
- 후면처리 입출력 : 파일로 저장 혹은 메일로 전달
  $ find . -name test.c -print > find.txt &
  $ find . -name test.c -print | mail chang &
  $ wc < inputfile &

  - 후면 처리 작업의 출력 결과를 조정하지 않으면 전면 처리 작업 화면 뒤섞이게 됨
  - 가급적 입출력을 파일로 받도록 설정

# 후면 처리 (jobs 명령 출력 항목)

- $ jobs (후면 실행 작업 목록 표시)

| 항목 | 출력예제 | 의 미 |
|---|---|---|
| 작업번호 | [1] | 작업번호로 백그라운드로 실행시킬 때마다 순차적으로 증가([1],[2],[3]...) |
| 작업순서 | +, - | 작업순서를 표시<br>• + : 가장 최근에 접근한 작업<br>• - : + 작업보다 바로 전에 접근한 작업<br>• 공백 : 그 외의 작업 |
| 상태 | 실행중 | 작업의 상태를 표시<br>• 실행중(Running) : 현재 실행중<br>• 완료됨(Done) : 작업이 정상적으로 종료<br>• 종료됨(Terminated) : 작업이 비정상적으로 종료<br>• 정지(Stopped) : 작업이 잠시 중단됨. |
| 명령 | sleep 100& | 실행중인 명령 |

# 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        //해당 작업만 listing
[1] + Running ( sleep 100; echo done )
```

# 후면 작업을 전면 작업으로 전환

전면 가는 후면 더 빠름

- 사용법

작업번호 안적으면 최근에 실행하던 작접을 전환

```
$ fg %작업번호
```

작업번호에 해당하는 후면 작업을 전면 작업으로 전환시킨다.

- 예

```
$ (sleep 100; echo DONE) &
[1] 10067
$ fg %1          // 1번 작업을 전면으로 전환시킴
( sleep 100; echo DONE )
```

# 전면 작업의 후면 전환: bg(background)

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

전면에서 실행 할 때

```
$ bg %작업번호
```

작업번호에 해당하는 중지된 작업을 후면 작업으로 전환하여 실행한다.

- 예

```
$ ( sleep 100; echo DONE )
^Z
[1]+ Stopped ( sleep 100; echo DONE )
$ bg %1
[1]+ ( sleep 100; echo DONE ) &
```

# 후면 작업의 입출력 제어

- 후면 작업의 출력
  - 후면 작업을 표준 출력으로 보내면 전면 작업과 출력이 섞이게 됨
  - 출력 재지정을 통해 사용

괄호가 없는 이유 하나로 봄 하나의 명령어
앞에서는 세미콜론 사용때문에(두가지 명령어로 봄) 괄호 사용

```
$ 명령어 > 출력파일 &
$ find . -name test.c -print > find.txt &
$ find . -name test.c -print | mail chang &
```

- 후면 작업의 입력
  - 키보드 입력은 전면 작업만 가능
  - 파일로부터 입력

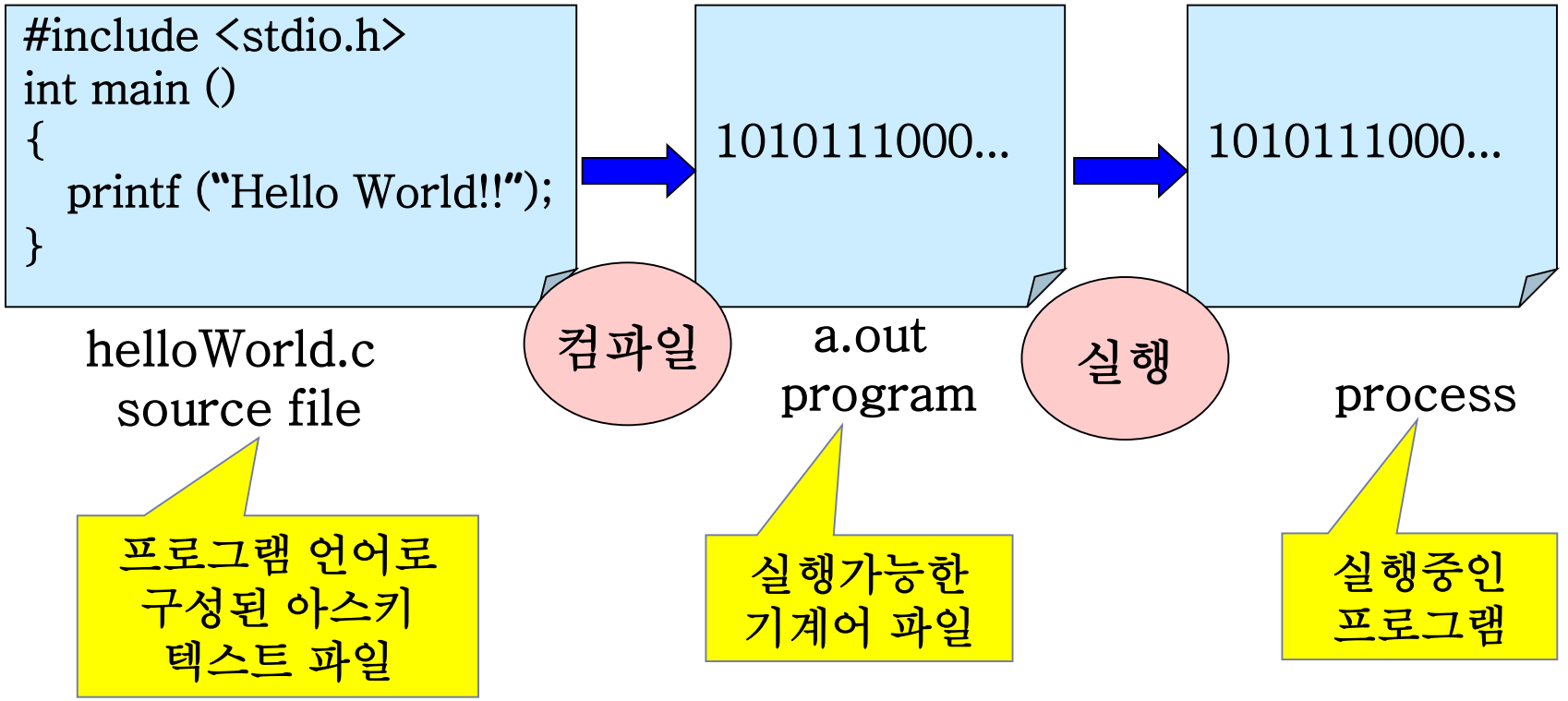
```
$ 명령어 < 입력파일 &
```

입력 재지정

여러 명령어 적을시 괄호 사용 해야 함

# 프로세스(process)

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



# 프로세스(process) 종류

Server Client
= daemon

| 종류 | 설명 |
|---|---|
| 데몬 (daemon) | UNIX커널에 의해 시작되는 프로세스로 서비스 제공을 위한 프로세스들이다. server |
| 부모 (parent) | 자식 프로세스를 만드는 프로세스 |
| 자식 (child) | 부모에 의해 생성된 프로세스로 실행이 끝나면 부모 프로세스로 돌아간다. |
| 고아 (orphan) | 자식프로세스가 종료하기 전에 부모가 종료된 프로세스. 고아프로세스는 1번 프로세스를 새로운 부모로 가진다. |
| 좀비 (zombie) | 부모프로세스에 의해 종료처리 되지 않은 (자식) 프로세스. 프로세스 테이블을 여전히 차지하고 있다 |

# 프로세스(process)

- ps 명령어를 사용하여 나의 프로세스들을 볼 수 있다.

```
$ ps
  PID   TTY    TIME     CMD
 8695 pts/3 00:00:00 csh
 8720 pts/3 00:00:00 ps

$ ps u          (u: 프로세스에 대한 보다 자세한 정보)
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
chang 8695 0.0  0.0  5252 1728 pts/3 Ss 11:12 0:00 -csh
chang 8793 0.0  0.0  4252 940  pts/3 R+ 11:15 0:00 ps u
```

VSZ (Virtual memory SiZe): 가상 메모리 크기 (태스크가 접근 가능한 모든 메모리 영역, 스와핑영역, 공유 라이브러리 영역 등)
RSS (Resident Set Size) : 실제 태스크에 할당된 물리 메모리 크기 (스위핑 영역 제외)

# (참고) STAT field of PS

```
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
chang 8695 0.0  0.0  5252 1728 pts/3 Ss 11:12 0:00 -csh
chang 8793 0.0  0.0  4252 940  pts/3 R+ 11:15 0:00 ps u
```

Stat 첫 번째 필드

| | |
|---|---|
| D | io와 같이 중지(interrupt)시킬 수 없는 잠자고 있는 (휴지) 프로세스 상태 |
| R | 현제 동작중이거나 동작할 수 있는 상태 |
| S | 잠자고 있지만, 중지시킬수 있는 상태 CPU 자원을 사용하지 않지만 다시 활성화 가능 |
| T | 작업 제어 시그널로 정지되었거나 추적중에 있는 프로세스 상태 |
| X | 완전히 죽어 있는 프로세스 |
| Z | 죽어 있는 좀비 프로세스 |

두 번째 필드

| | |
|---|---|
| < | 프로세스의 우선 순위가 높은 상태 |
| N | 프로세스의 우선 순위가 낮은 상태 |
| L | 실시간이나 기존 IO를 위해 메모리 안에 잠겨진 페이지를 가진 상태 |
| s | 세션 리터(주도 프로세스) |
| I | 멀티 쓰레드 |
| + | 포어그라운드 상태로 동작하는 프로세스 |

사용자가 직접 실행하고 제어하는 작업이
현재 터미널에서 활성화된 상태

# 프로세스 상태: ps

telnet hanbit.co.kr

```
$ ps -ef | more          //System V 기준
     UID   PID    PPID   C    STIME    TTY      TIME   CMD
    root     0       0   0    3월 04     ?     0:00   sched
    root     1       0   0    3월 04     ?     0:53   /etc/init -
   user1 25786   25600   0  10:30:55  pts/6     0:00    vi prog3.c
   user2 25600   25598   0  10:03:51  pts/6     0:00   -ksh
```

| 구분 | 설명 | 구분 | 설명 |
|---|---|---|---|
| UID | 소유자의 사용자 ID | STIME | 프로세스 시작시간 |
| PID | 프로세스 번호 | TTY | 터미널 번호( ? : 데몬) |
| PPID | 부모 프로세스 번호 | TIME | CPU 사용시간 |
| C | 프로세스 우선순위 | CMD | 명령어 이름 |

# ps

$ ps aux (aux : 시스템 내의 모든 프로세스에 대한 자세한 정보)

```
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 -csh
chang 8976 0.0 0.0 4252 940 pts/3 R+ 11:24 0:00 ps aux
```

# ps

```
yu22312072@acslab-146:~$ ps -u
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
yu22312+ 2506355  0.0  0.0  12580  5760 pts/2    Ss   16:11   0:00 -bash
yu22312+ 2507471  0.0  0.0  13984  3712 pts/2    R+   16:36   0:00 ps -u
yu22312072@acslab-146:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
yu22312+ 2506355 2506354  0 16:11 pts/2    00:00:00 -bash
yu22312+ 2507478 2506355  0 16:36 pts/2    00:00:00 ps -f
```

파이퍼 있어도 되고 없어도 됨

- $ ps -aux (BSD 유닉스)
  - - a: 모든 사용자의 프로세스를 출력
  - - u: 프로세스에 대한 좀 더 자세한 정보를 출력
  - - x: 더 이상 제어 터미널을 갖지 않은 프로세스들도 함께 출력
- $ ps -ef (시스템 V)
  - - e: 모든 사용자 프로세스 정보를 출력
  - - f: 프로세스에 대한 좀 더 자세한 정보를 출력

계열 차이

u, f 정보가 다름

```
yu22312072@acslab-146:~$ ps -e | head -5
    PID TTY          TIME CMD
      1 ?        02:48:51 systemd
      2 ?        00:00:03 kthreadd
      3 ?        00:00:00 pool_workqueue_release
      4 ?        00:00:00 kworker/R-rcu_g
yu22312072@acslab-146:~$ ps -a | head -5
    PID TTY          TIME CMD
2292169 pts/41   00:00:00 zsh
2292170 pts/41   00:00:04 zsh
2292173 pts/41   00:00:00 zsh
2292178 pts/41   00:00:17 gitstatusd-linu
```

- ps 명령어는 시스템에 따라 옵션 및 사용 방법의 차이가 크다

# 특정 프로세스 리스트: pgrep

- 특정 프로세스만 리스트

  $ ps –ef | grep –w sshd  //sshd 프로세스만 조회하는 기존의 방법

  shell daemon

- 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.
```

추가 명령어로서 리눅스 종류와 버전에 따라 해당 명령어가 없을 수 있음

# 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

# kill 명령어

- 프로세스를 강제적으로 종료시키는 명령어

  $ kill [시그널] 프로세스번호 //pid 사용

  $ kill %작업번호 // job id 사용

  $ kill 8320 혹은

  $ kill %1

  [1] Terminated ( sleep 100; echo done )

  프로세스 관련 명령어들 :

  명령어 &, ^c, ^z, bg, jobs, fg %1, kill %1, kill 8320...

# kill 명령어

- Kill 명령어는 프로세스에게 시그널을 보내는 명령어
- 시그널(signal)이란?
  - 프로세스에게 보내는 신호
  - 프로세스는 이 신호에 응답한다.
  - 응답의 종류 : 신호 무시, 프로세스 종료 등
  - kill 명령으로 신호를 보낸다.

해보기

  - $ man signal 로 자세한 정보를 찾아볼 수 있다.
  - 혹은 kill –l : 시그널 목록 확인
  - 시그널 -9 : 강제 종료 시그널

# kill 명령어

- 대표적인 시그널의 종류

| 시그널 번호 | 시그널 이름 | 기능 | 기본 응답 |
|---|---|---|---|
| 1 | SIGHUP | • 터미널 연결이 끊어진 경우에 발생 | 종료 |
| 2 | SIGINT | • 보통 Ctrl-C에 의해 발생 | 종료 |
| 9 | SIGKILL | • 프로세스를 kill 시킨다.<br>• 이 시그널은 무시할 수 없다. | 종료 |
| 15 | SIGTERM | • 프로세스를 종료시킨다.<br>• 이 시그널은 무시할 수도 있다.<br>• kill 명령이 보내는 default 시그널 | 종료 |

# sleep

- sleep 명령어
  - 지정된 시간만큼 실행을 중지한다.

$ sleep 초

$ (echo 시작; sleep 5; echo 끝)  // ( ) 는 명령어 묶음 기능 제공
// 여기서는 없어도 됨

백그라운드 실행 할 때

# exit

exit(0) = return(0)
정상종료

- exit
  - 쉘을 종료하고, 즉 로그 아웃 후 종료코드(exit code)를 부모 프로세스에 전달

$ exit [종료코드]

부모 프로세스는 누구인가?

만약 내가 여러 개의 쉘을 사용하고 있었다면?

# 핵심 개념

- 유닉스의 디렉토리는 루트로부터 시작하여 계층구조를 이룬다.
- 절대 경로명은 루트 디렉토리부터 시작하고 상대 경로명은 현재 디렉토리부터 시작한다.
- 파일의 사용권한은 소유자, 그룹, 기타로 구분하여 관리한다.
- 출력 재지정은 표준출력 내용을 파일에 저장하고 입력 재지정은 표준입력을 파일에서 받는다.
- 실행중인 프로그램을 프로세스라고 한다.

# (참고) 리눅스 활용을 위한 여러 명령 그룹들

- Shell / Shell script
  - Bourne shell, C shell, Korn shell, Bash shell
  - Shell script programming
- Linux User Command
  - file/directory management (ls, cp, mkdir, rm, ln)
  - permission control (chmod, umask)
- System Administration Command
  - User management (useradd/usermod/userdel)
  - Disk management (sync, du, df, mound)
  - su, sudo, RPM management, apt-get (advanced packaging tool, ubuntu package manager)
- Network Configuration Command
  - ifconfig, netstat, traceroute, route
- Security-related Command
  - Password control, log file management, internet service control
- System Programming Command and Utility (Linux programming)
  - vi, gcc, make, gdb
  - Assembler (as), linker, loader (ld)
  - static library (ar), shared object