redhat CentOS

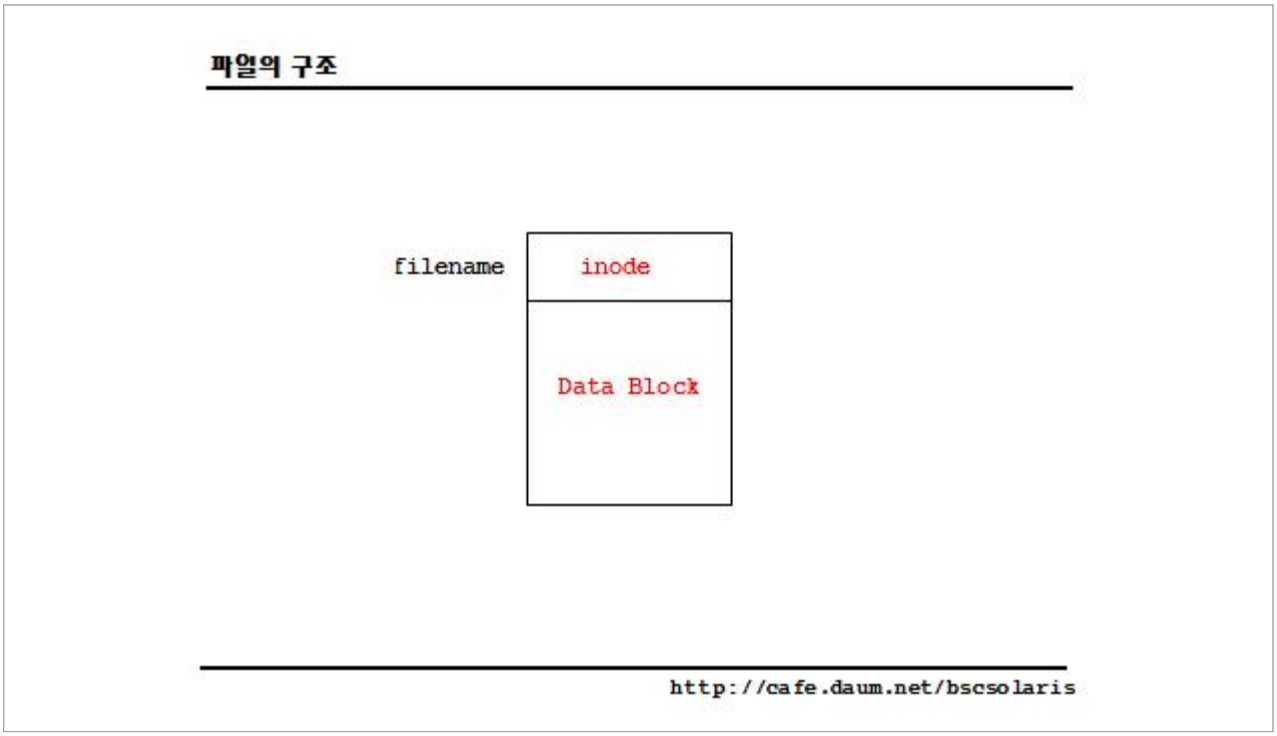
# Unit 6
# File Type

http://cafe.daum.net/bscsolaris

**단원 목표**

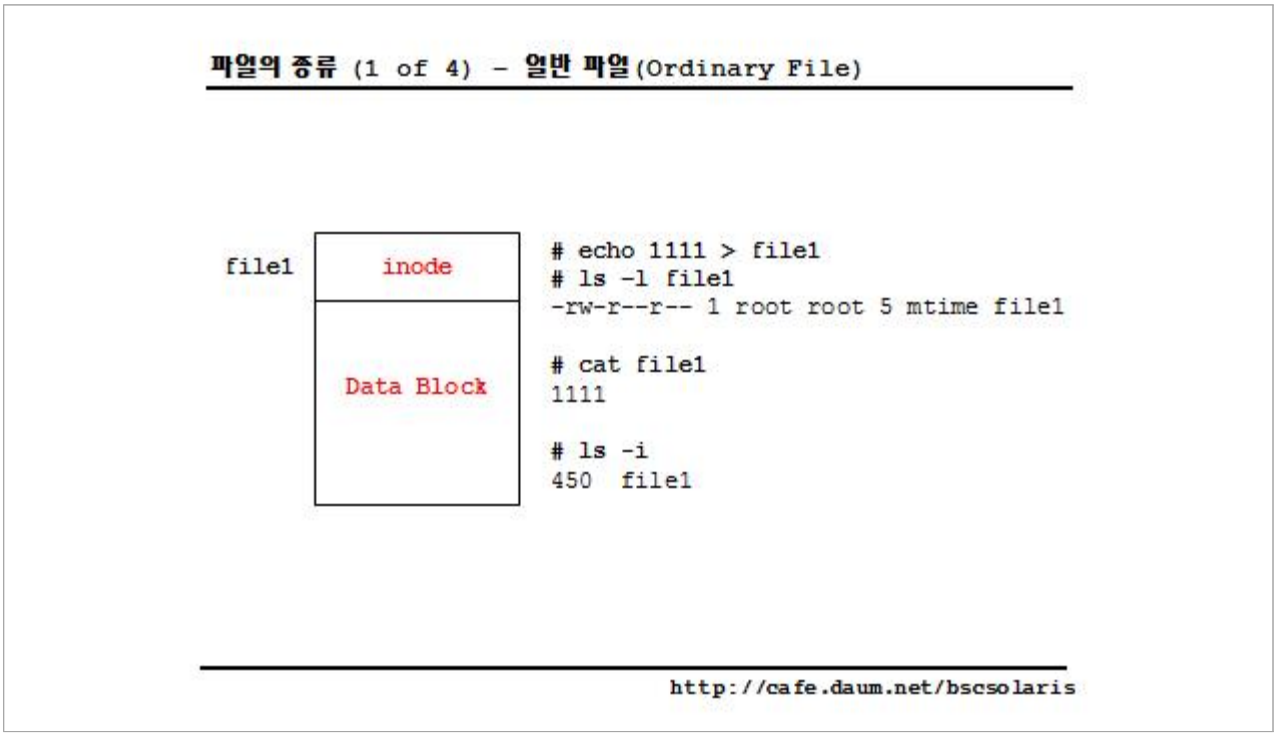
- 파일의 구조
- 일반 파일(Odinary File)
- 디렉토리 파일(Directory File)
- 링크 파일(Link File)
- 디바이스 파일(Device File)

http://cafe.daum.net/bscsolaris




■ 파일(File)의 종류
- ■ 일반 파일(Egular File)
- ■ 디렉토리 파일(Directory File)
- ■ 링크 파일(Link File)
- ■ 장치 파일(Device File)
- ■ 소켓 파일(Socket File)
- ■ 도어 파일(Door File) - 현재 리눅스에서는 사용하지 않음
- ■ 파이프 파일(Pipe File)

파일과 디렉토리의 내용들을 다루기 위해서는 위와 같은 기본적인 명령어에 익숙해져야 한다. 리눅스 시스템에서 파일의 구조(File Structure) 파일이름, Inode (Index Node)와 데이터 블럭(Data block)로 구분할 수 있다. 파일의 이름은 유일해야 하고 대소문자는 다른 문자로 인식하게 된다. Inode는 파일의 속성정보와 데이터 블럭을 포인트하는 정보가 들어 있으며 ls 명령어에 -l 옵션을 사용하여 대부분의 정보를 확인 할 수 있다. 데이터 블록안에는 실제 파일에 저장되는 내용이 들어가게 된다. 일반파일, 디렉토리, 심볼릭링크 파일들은 모두 하나이상의 데이터 종류를 저장한다. 하지만 디바이스 파일은 데이터를 저장하지 않는다. 대신에 디바이스 파일은 디바이스 제어접근(Access)을 제공한다.




## 1 일반파일 (Regular File)

리눅스에서는 모든 것을 파일로 다룬다. 일반 파일, 디렉토리 파일, 링크 파일등이 있다. 파일의 종류에 관한 정보는 ls 명령어의 -l 옵션을 사용하여 출력되는 화면에 첫 번째 문자에서 확인이 가능하다. 또한 file 명령어에 파일의 이름을 인자로 받아서 파일의 종류를 확인 할 수도 있다.

| 문자 | 파일 종류 |
|---|---|
| - | 일반 파일(Egular File) |
| d | 디렉토리 파일(Directory File) |
| b | 블록 디바이스 파일 (Block Device File, (예) /dev/sha, /dev/hda, /dev/fd0) |
| c | 문자 디바이스 파일 (Character Device File, 입출력 장치) |
| l | 심볼릭 링크(Symbolic Link File) |

| [참고] 자주 사용되는 기본 명령어 |
|---|
| • 디렉토리에 관련된 기본 명령어: ls, pwd, mkdir, cd, mv 등<br>• 파일과 관련된 기본 명령어: cat, more, cp, rm, head, tail 등<br>• 파일과 디렉토리 검색에 관련한 기본 명령어: grep, find 등 |

### (1) 파일의 이름(filename)

리눅스 시스템에서 파일의 이름은 파일을 접근하기 위해 사용한다. 파일은 Inode와 함께 파일의 구성요소로서 같은 디렉토리 안에서 파일의 이름은 유일한 것이어야 한다. 예를 들어 현재 디렉토리 밑에 존재하는 디렉토리와 같은 이름은 일반파일을 생성 할 수는 없다.

[참고] 파일의 이름 생성 시 규칙(filename convention)

- 파일의 이름은 최대 255자까지 생성이 가능하다.
- 파일의 이름은 문자와 숫자를 사용 할 수 있으며 대소문자를 구별한다.
- 파일의 이름으로 .(Dot), _(Underbar), -(Dash) 사용가능하다.
  (예) file1.txt backup-2024-0725.bk
- 파일의 이름으로 특수문자(Metacharater)를 사용하면 안된다.
  대표적인 특수문자는 *(Aterisk), /(Slash), ₩(Back Slash), ,(Comma), '(Single Quote), "(Double Quote) (Semicolon), &(Ampersand), |(Pipeline), <(Angle Blaket)등이다.
- ".(Dot)"로 시작하는 파일의 이름은 시스템의 환경파일들이므로 일반파일의 이름으로 권장하지 않는다.
  (예) ~/.bashrc
- 파일의 이름에 공백이나 탭을 사용하는 것을 권장하지 않는다.

### (2) Inode

Inode는 파일에 대한 정보를 담고 있는 부분이다. 일반적으로 Inode에는 크게 두가지 부분을 포함하고 있다. 첫 번째는 파일에 대한 속성정보(Ownership, Groupship, File Permission Mode등)와 데이터 블록을 가리키고 있는 포인터(Direct/Indirect Pointer)이다. Inode는 숫자로 되어져 있으며 각 파일 시스템은 자신의 Inode 테이블(Inode Tables)을 가지고 있다. 파일이 새로운 파일시스템이 만들어지는 경우 파일시스템 안에서는 새로운 Inode 번호을 할당 받게 된다.

[참고] Inode의 정보

- 파일의 종류(File Type)
- 파일의 퍼미션모드(File Permission Mode)
- 파일의 소유자, 그룹(File Ownership, Groupship)
- 파일의 링크수(Hard Link Count)
- 파일의 마지막 접근 시간(Access Time), 수정시간(Modification Time), 수정시간(Changed Time)
- 파일의 크기(Bytes, 할당된 또는 사용중인 데이터블럭의 수)
- 두가지 형태의 포인터(Directe Pointers and Indirect Pointers)

### (3) Data Block

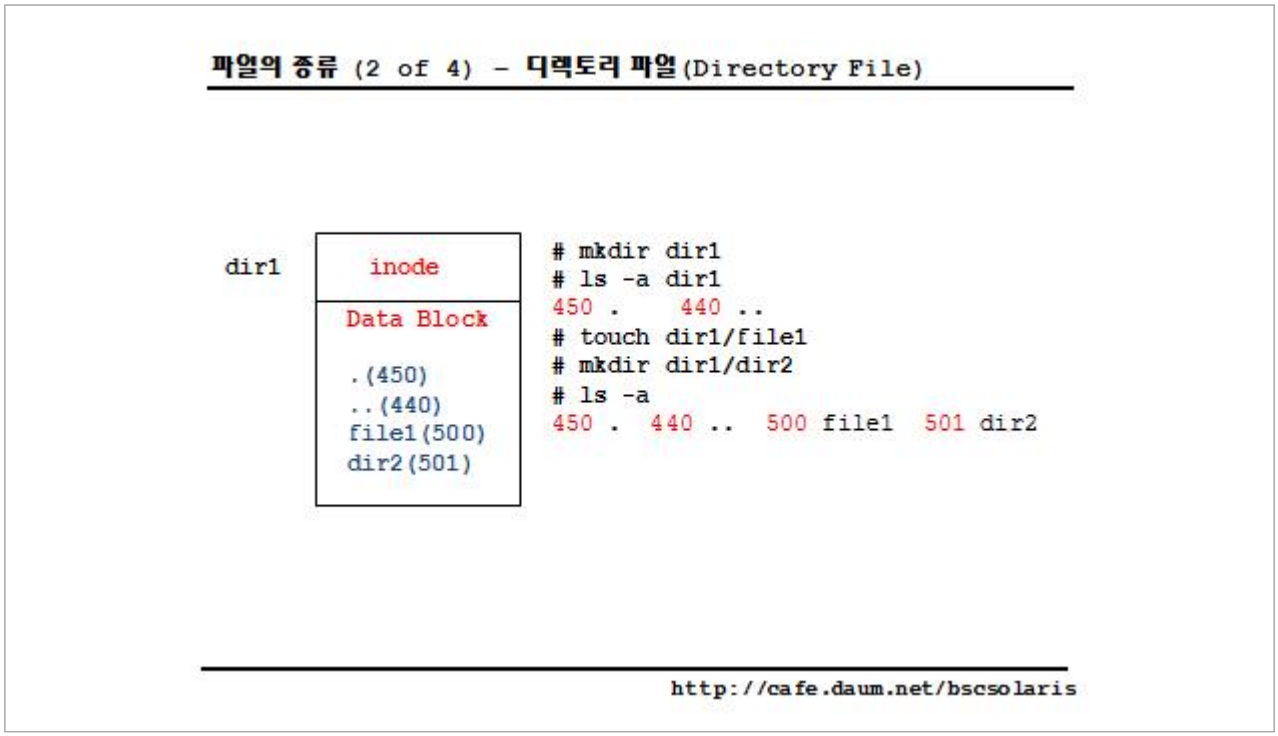
데이터블럭은 디스크 공간에 대한 단위(Units of Disk Space)로서 데이터를 저장하는 역할을 가진다. 일반파일, 디렉토리, 심볼릭 파일들은 데이터 블록을 사용하지만 일반파일과 다른 구조를 가지고 있는 디바이스 파일은 데이터 블럭에 데이터를 저장하지 않고 주 디바이스 숫자(Major Device Number)와 부 디바이스 숫자(Minor Device Number)를 담고 있다.

[참고] 데이터 블럭(Data Block)의 내용

- 일반파일의 경우 파일의 내용이 들어 있다.
- 디렉토리인 경우 안에 포함된 파일과 디렉토리이름이 들어 있다.

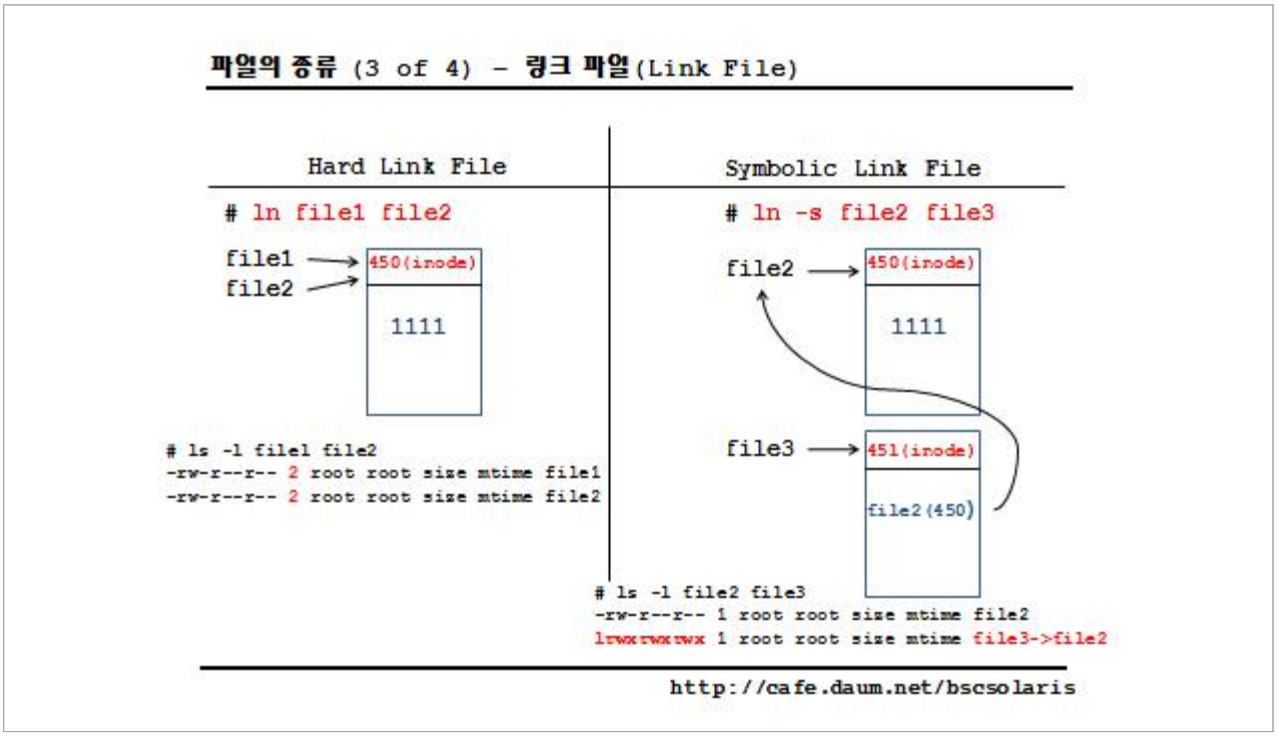
일반파일은 리눅스에서 찾을 수 있는 거의 대부분의 파일종류이다. 데이터 블록에 들어가는 데이터는 많은 형태는 ASCII (American Standard Code for Information Interchange) 텍스트, 바이너리 데이터, 이미지 데이터, 데이터베이스 데이터, 애플리케이션 데이터 등이 있다. 일반파일을 만드는 방법 또한 많다.

예를 들어 vi 편집기를 사용 할 수도 있고, 컴파일을 통해 바이너리를 생성 할 수도 있고, touch 명령어를 통해 빈 파일을 생성 할 수도 있다.




## 2 디렉토리 파일 (Directory File)

디렉토리의 데이터 블록 안에는 실제 디렉토리가 포함하고 있는 파일과 디렉토리에 대한 정보를 가지고 있다. 파일의 이름, 디렉토리의 이름 Inode 번호를 저장하고 있다. 디렉토리의 데이터 블록 안에는 어떤 파일 종류이든 파일의 종류에 맞는 정보가 들어 갈수 있다.




## 3 링크 파일 (Link File)

■ 링크 파일
- 하드 링크 파일(Hard Link File)
- 심볼릭 링크 파일(Symbolic Link File), 소프트 링크 파일(Soft Link File)

### (1) 하드 링크 (Hard link)

원본파일의 경로를 저장하고 있는 파일을 말한다. 파일을 실제 경로가 아니라 사용하기 편리한 다른 경로로 접근할 수 있도록 지정하는 명령어이다. 링크파일의 type은 "ls -l" 했을 때 맨 앞의 글자가 "l"로 표시되어 있다.

하드 링크는 똑같은 파일크기로 원본 파일이 수정될 경우, 하드 링크된 파일도 원본과 동일하게 변경되며 항상 같은 내용을 유지할 수 있다. 원본이 삭제되어도 원본과 동일한 내용을 가지고 있으므로 자원을 공유하되 데이터를 안전하게 관리하고자 할 때 유용하게 사용할 수 있다. 하드링크는 디렉토리에는 만들 수 없다.

하드 링크는 파일명이 하나 더 만들어져 동일한 I-node 번호를 가르치게 된다. 하드 링크는 같은 I-node 번호를 저장하고 있으므로 원본 파일을 지워도 원본파일의 파일명만 지워질 뿐 inode가 지워지는 것이 아니기 때문에 링크로 inode값을 가르키는 링크 파일에 영향을 미치지 않는다.

원본 파일과 링크 파일의 권한은 항상 같다. 그 이유는 I-node에 권한이 저장이 되어 있기 때문이다. 즉 파일명에 대해서 권한을 부여하는 것이 아니고 inode번호에 대해 권한을 설정하기 때문이다. 이는 각기 다른 이름을 가진 하드링크의 파일명에 대한 권한을 변경해줘도 파일명에 권한 설정이 되는 것이 아니기 때문에 서로 다른 이름을 가진 하드링크 파일의 권한은 동일하게 유지가 되는 것이다.

하드 링크에는 다음과 같은 한계가 있다.
(ㄱ) 하드링크는 일반 파일에서만 사용할 수 있다. 디렉토리 또는 특수 파일등에 대한 하드링크를 불가능하다.
(ㄴ) 하드링크는 두 파일이 같은 파일시스템에 있는 경우에만 사용할 수 있다.
따라서, 위와 같은 경우라면 심볼릭 링크를 사용하면 된다.

### (2) 심볼릭 링크, 소프트 링크 (Symbolic link, soft link)

일반적으로 링크라고 하면 심볼릭 링크를 말한다. 심볼릭 링크는 소프트 링크라고도 하며 불필요한 파일의 복사를 하지 않아도 된다. 보통 여러 디렉토리에서 동일한 라이브러리를 요구할 경우나, 하나의 파일을 여러 사람이 공통으로 사용할 경우도 많이 쓴다. 소프트 링크의 퍼미션은 모든 유저에게 모든 권한(rwxrwxrwx=777)을 준다.

포인트 하는 정보가 들어 있을 뿐 원본파일은 아니다. 심볼릭 size는 포인트 정보만 들어 있어 cat으로 보여지는 정보가 많던 적던간에 동일한 size를 갖는다. 따라서 윈도우에서 바탕화면에 바로가기가 되어있어도 많은 양의 리소스를 차지하지 않고 포인트 정보를 담은 size만이 disk의 공간을 차지하게 된다.

원본파일이 지워지게 되면 링크 파일에 영향을 미치게 된다. 소프트링크 파일의 권한은 항상 모든 권한이다.

**[참고] 소프트 링크파일의 퍼미션은 항상 777인 이유는?**

소프트링크를 만드는 이유는 모든 사용자가 자원을 공유하려 할 때 많이 사용하기 때문이다. 그런데 소프트 링크파일은 그 안의 데이터 블록에 원본파일의 inode의 경로를 포함하는 것이므로 링크파일의 퍼미션이 모든 권한이라 하더라도 실제로는 원본 파일의 권한을 따르게 된다. 링크파일의 퍼미션을 변경하면 원본 파일의 퍼미션이 변경이 된다.

```
# cd /etc
# ls -l grub2.cfg
lrwxrwxrwx. 1 root root 22 Apr 14  2020 grub2.cfg -> ../boot/grub2/grub.cfg

# ls -l /boot/grub/grub.conf
-rw-r-----. 1 root root 5.1K Feb  6 14:23 /boot/grub2/grub.cfg

# su - fedora
$ cat /etc/grub2.cfg
cat: /etc/grub2.cfg: Permission denied
```

**[명령어 형식]**

(하드링크 )# ln file1 file2
(심볼릭링크)# ln -s file1 file2

**[명령어 옵션]**

| 옵 션 | 설 명 |
|---|---|
| -b, --backup | 대상 파일이 있다면 백업파일을 생성한다. |
| -f, --force | 링크를 생성할 대상 파일이 있더라도 강제적으로 새로운 링크를 생성한다. |
| -i, --interactive | 링크를 생성할 대상 파일이 있을 경우, 삭제 유무를 사용자에게 물어 본다. |
| -n, --no-dereference | 링크할 원본이 심볼릭 파일이면, 그 심볼릭 파일의 대상 파일을 추적하여 링크한다. |
| -s, --symbolic | 링크할 원본이 심볼릭 파일이면, 심볼릭 파일을 링크한다. |
| -S, --suffix backup-suffix | 링크를 생성할 대상 파일이 이미 있을 경우, 이전의 대상파일을 백업할 파일의 확장자를 지정한다. |

[실습] 링크 실습

■ 사용 시스템
- server1

■ 실습 시나리오
- 하드 링크(Hard Link) 기본에 대한 실습
- 하드 링크(Hard Link) 실습
- 심볼릭 링크(Symbolic Link) 실습

[EX1] **하드링크** 기본에 대한 실습 - 디렉토리 하드링크 수에 대한 실습
- 하드링크는 기본적으로 사용되는것이다.

```
# touch file1
# ls -l file1
-rw-r--r-- 1 root root 0 Aug 13 09:47 file1

# mkdir dir1
# ls -ld dir1
drwxr-xr-x 2 root root 4.0K Aug 13 09:48 dir1/
```

- [파일]에 대한 하드링크 수는 1이다.
  -> [파일의 이름]과 [Inode] 하나가 매핑(mapping)이 되어 있으면 하드링크 1 이다.
- [디렉토리]에 대한 하드링크 수는 2이다.
  -> 디렉토리 안에 들어 있는 디렉토리의 개수가 디렉토리에 대한 하드링크 수이다.

① 실습용 위한 준비 - 디렉토리 하드링크 수에 대한 개념 확인
```
[root@server1 /test]# cd /test
[root@server1 /test]# rm -rf /test/*
```

② dir1 디렉토리 생성 및 하드 링크 수 변화 확인
```
[root@server1 /test]# mkdir dir1
[root@server1 /test]# ls -l            <---- dir1 하드링크 수 :
```

③ dir1/dir2 디렉토리 생성 및 하드 링크 수 변화 확인
```
[root@server1 /test]# mkdir dir1/dir2
[root@server1 /test]# ls -l            <---- dir1 하드링크 수 :
```

④ dir1/dir3 디렉토리 생성 및 하드 링크 수 변화 확인
```
[root@server1 /test]# mkdir dir1/dir3
[root@server1 /test]# ls -l            <---- dir1 하드링크 수 :
```

⑤ dir1/file1 일반 파일 생성 및 하드 링크 수 변화 확인
```
[root@server1 /test]# touch dir1/file1
[root@server1 /test]# ls -l            <---- dir1 하드링크 수 :
```

⑥ dir1/dir2/dir4 디렉토리 생성 및 하드 링크 수 변화 확인
```
[root@server1 /test]# mkdir dir1/dir2/dir4
[root@server1 /test]# ls -l            <---- dir1 하드링크 수 :
```

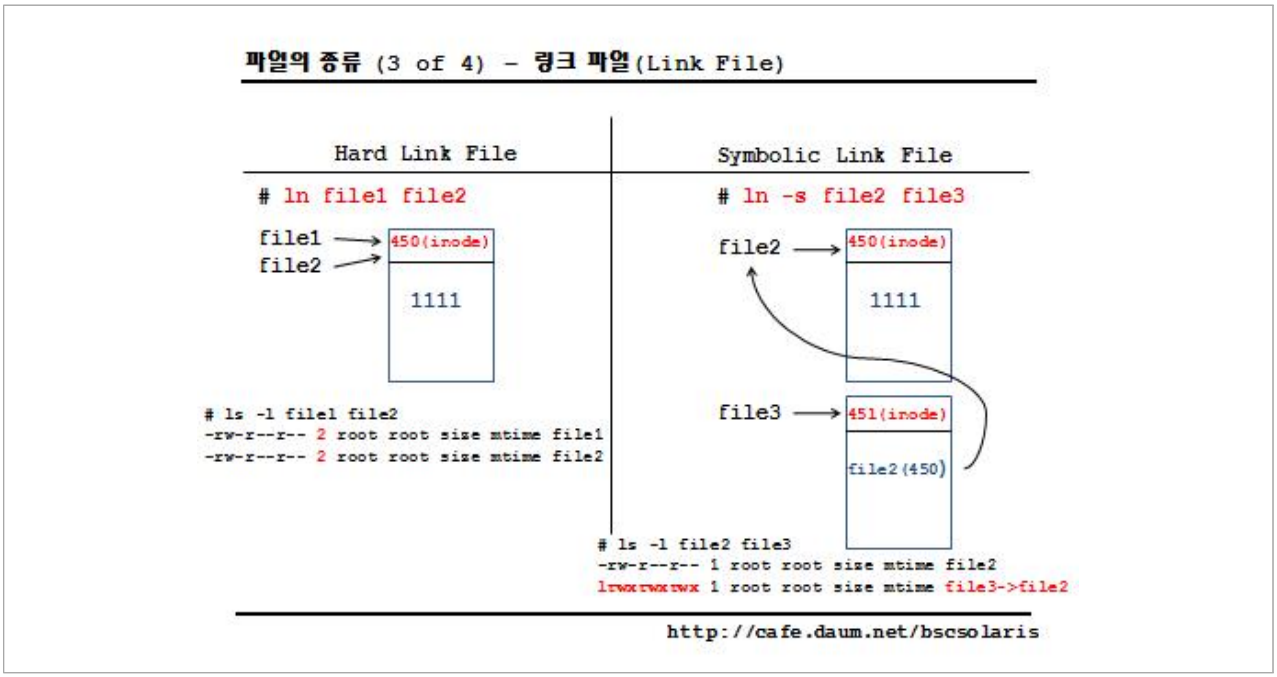
⑦ /(root) 디렉토리 안에 많은 디렉토리에 대한 하드 링크 수 확인
```
[root@server1 /test]# ls -l /
```

```
drwxr-xr-x  25 root root 4.0K Aug 10 20:48 var/
```

- 여러가지 출력내용 중 /var 디렉토리에 대한 내용을 확인한 것이다.

[EX] 하드링크 실습 - 하나의 파일에 대한 다른 파일을 하드링크 거는 실습

■ 하드 링크(Hard Link)




① 실습을 위한 준비

```
[root@server1 /test]# cd /test && rm -rf /test/*
[root@server1 /test]# echo 1111 > file1
[root@server1 /test]# ls -l file1
```

```
-rw-r--r-- 1 root root 5 Feb 17 10:04 file1
```

```
[root@server1 /test]# cat file1
```

```
1111
```

② 하드 링크 걸기 및 상태 확인

```
[root@server1 /test]# ln file1 file2
[root@server1 /test]# ls -li
```

```
35863700 -rw-r--r-- 2 root root 5 Feb 17 10:04 file1
35863700 -rw-r--r-- 2 root root 5 Feb 17 10:04 file2
```

- 특이사항: 하드 링크 수가 '2'가 되었다.

③ file2 내용을 추가한 후 file1 파일의 변화 확인

```
[root@server1 /test]# echo 2222 >> file2
[root@server1 /test]# cat file1
```

```
1111
2222
```

④ file1 삭제한 후 file2 파일의 변화 확인

```
[root@server1 /test]# rm -f file1
[root@server1 /test]# cat file2
```

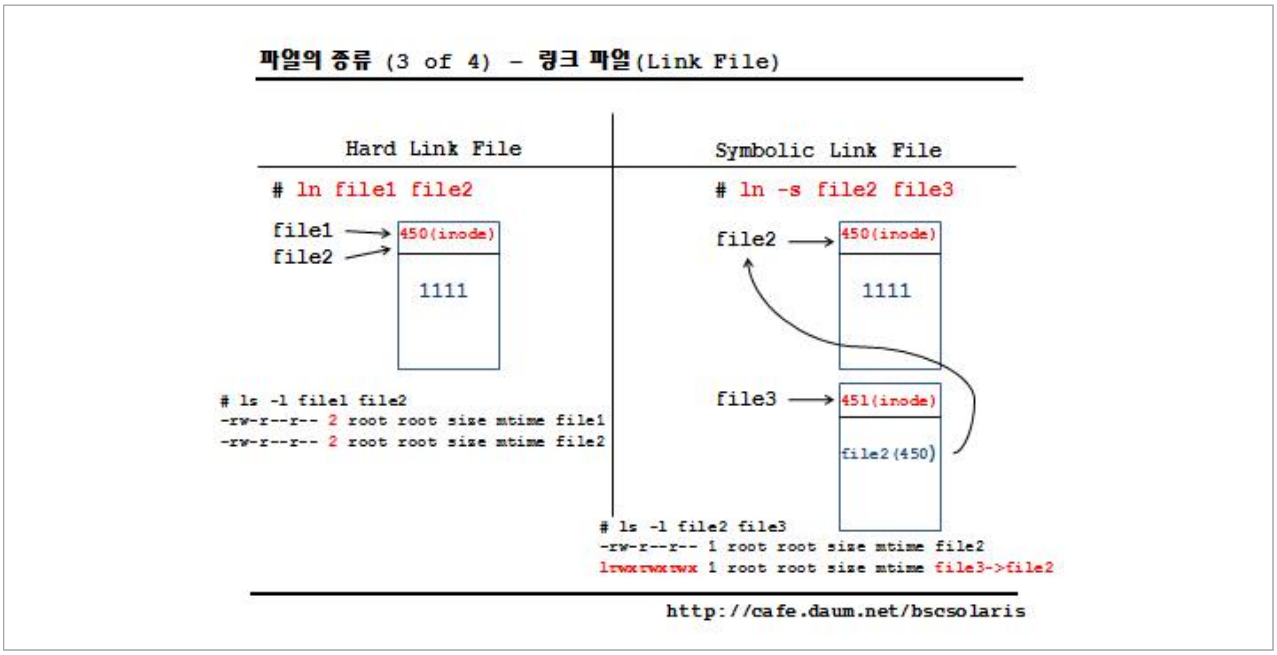
```
1111
2222
```

```
[root@server1 /test]# ls -l
```

```
-rw-r--r-- 1 root root 10 Feb 17 10:06 file2
```

[EX] 실볼릭 링크 실습 - 하나의 파일에 대한 다른 파일을 심볼릭 링크 거는 실습

## ■ 심볼릭 링크(Symbolic Link)




① file2 파일에 file3 파일을 심볼릭 링크 걸기 및 상태 변화 확인

```
[root@server1 /test]# ln -s file2 file3
[root@server1 /test]# ls -li
```

```
35863700 -rw-r--r-- 1 root root 10 Feb 17 10:06 file2
35863701 lrwxrwxrwx 1 root root  5 Feb 17 10:08 file3 -> file2
```

- 파일 특성:
  * 파일 종류: l
  * 파일 퍼미션: rwxrwxrwx
  * 파일 이름: file3 -> file2

② file3 파일에 내용을 추가한 후 file2 파일의 변화 확인

```
[root@server1 /test]# echo 3333 >> file3
[root@server1 /test]# cat file2
```

```
1111
2222
3333
```

③ file2 파일 삭제 후 file3의 상태 변화 확인

```
[root@server1 /test]# rm -f file2
[root@server1 /test]# cat file3
```

```
cat: file3: No such file or directory
```

[EX3] **하드 링크** vs **심볼릭 링크**
- 하드 링크와 심볼릭 링크의 특성에 차이점에 대해서 확인한다.

■ 하드링크(**# ln file1 file2**)
- 두개의 파일의 inode 번호가 동일한가?
- ls -li 특이한 변화는?
- 두개의 파일 사이즈는 동일한가?
- file2의 편집하면?

■ 심볼릭 링크(**# ln -s file1 file2**)
- 두개의 파일의 inode 번호가 동일한가?
- ls -li 특이한 변화는?
- 두개의 파일 사이즈는 동일한가?
- file2의 편집하면?

[참고] "리눅스 심볼릭 링크 파일"과 "윈도우 바로가기 아이콘("아이콘".lnk)" 비교

***NIX 심볼릭 링크 == 윈도우 [바로가기] 아이콘**

윈도우의 바로가기 기능은 OS 차원의 기능이고, 심볼릭 링크는 파일시스템 차원의 기능이어서 그 기능의 동작 방식부터가 다르다. 윈도우의 바로가기 기능의 경우 바로가기 파일을 통해 실현 가능한데, 이 파일은 일반 파일과 동일하게 메타데이터와 MFT 엔트리를 가지고 있고, 심볼릭 링크의 경우 원본 파일을 가리키고 있는 것은 파일이 아닌것으로 파일시스템이 인식하기 때문에 이 두가지은 엄밀히 다르다. 하지만, 일반적인 특성이 같기 때문에 비슷한 기능으로 봐야 할것이다.

(실무 예) **[하드 링크]** & **[심볼릭 링크]**의 차이점
하드링크는 같은 파일시스템내에서만 링크를 걸수가 있다. 하지만 심볼릭 링크는 다른 파일시스템에도 링크를 걸수가 있다. 또한, 하드링크는 디렉토리에 다른 파일을 링크를 걸수는 없다. 이유는 디렉토리의 하드링크는 디렉토리 안에 들어 있는 디렉토리의 목록이기 때문이다. 하지만 심볼릭 링크는 하드링크와는 다르게 디렉토리를 파일에 링크로 걸수 있다.
- (ㄱ) 파일시스템을 넘어서 링크를 걸수 있는가?
- (ㄴ) 디렉토리에 링크를 걸수 있는가?

```
# ln -s /var /test/var
# cd /test/var
# ls
# ls /var
# cd
```

[EX4] 시스템에서 사용되는 심볼릭 링크 확인

* /bin -- symbolic link --> /usr/bin
* /sbin -- symbolic link --> /usr/sbin

① /bin 디렉토리 정보 확인
**[root@server1 ~]# ls -l /bin**

```
lrwxrwxrwx. 1 root root 7 Jun 25 23:23 /bin -> usr/bin/
```

② /bin vs /usr/bin 디렉토리 정보 확인
**[root@server1 ~]# cd /bin**
**[root@server1 /bin]# ls**
-> 출력 내용 확인

**[root@server1 /bin]# cd /usr/bin**
**[root@server1 /usr/bin]# ls**
-> 출력 내용 확인

③ /sbin 디렉토리 정보 확인
**[root@server1 /usr/bin]# ls -l /sbin**

```
lrwxrwxrwx. 1 root root 8 Jun 25 23:23 /sbin -> usr/sbin/
```

④ /sbin vs /usr/sbin 디렉토리 정보 확인
**[root@server1 /usr/bin]# cd /sbin**
**[root@server1 /sbin]# ls**
-> 출력 내용 확인

**[root@server1 /sbin]# cd /usr/sbin**
**[root@server1 /usr/sbin]# ls**
-> 출력 내용 확인

[EX5] Linux 시스템 사용되고 있는 Symbolic Link 예
- (선수작업) # yum install httpd mod_ssl -y
- **"systemctl enable|disable httpd.service"** 정확인 의미가 무엇인가?

**[root@server1 ~]# systemctl enable httpd**

```
Created symlink /etc/systemd/system/multi-user.target.wants/httpd.service →
/usr/lib/systemd/system/httpd.service.
```

- 부팅시에 httpd.service 서비스를 기동시키겠다.

**[root@server1 ~]# ls -l /etc/systemd/system/multi-user.target.wants/httpd.service**

```
lrwxrwxrwx 1 root root 37 Aug 21 12:43 httpd.service -> /usr/lib/systemd/system/httpd.service
```

**[root@server1 ~]# systemctl disable httpd**

```
Removed /etc/systemd/system/multi-user.target.wants/httpd.service.
```

**[root@server1 ~]# ls -l /etc/systemd/system/multi-user.target.wants/httpd.service**

```
ls: cannot access 'httpd.service': No such file or directory
```

**[root@server1 ~]# systemctl enable httpd**

```
Created symlink /etc/systemd/system/multi-user.target.wants/httpd.service →
/usr/lib/systemd/system/httpd.service.
```

**[root@server1 ~]# ls -l /etc/systemd/system/multi-user.target.wants/httpd.service**

```
lrwxrwxrwx 1 root root 37 Aug 21 14:18 httpd.service -> /usr/lib/systemd/system/httpd.service
```

(실무 예) 심볼릭 링크는 일반적으로 상대경로 사용하지 않는다.
- 일반적으로 심볼릭 링크는 특별하지 않은 이상 절대경로를 사용하여 거는 것이 좋다.
- 하지만, 일부로 상대경로를 사용하는 경우도 존재한다.
- 권장하는 방식은 절대경로를 사용하는 방식이다.

(권장 X) # ln -s dir1 dir2
(권장 O) # ln -s /test/dir1 /test/dir2

(실무 예) 웹서버(WAS 서버)에서 웹소스 디렉토리 마이그레이션 하는 작업
- 웹서버의 소스 코드가 존재하는 디렉토리(또는 파일시스템)의 마이그레이션 작업을 진행하는 작업은 시간이 많이 소요되는 작업이다.
- 하지만, 개발자가 바로 작업을 진행해 달라고 하면, 관리자 입장에서는 바로 진행할 수 있도록 하기 위해서 임시적인 방법으로 심볼릭 링크를 사용하여 작업을 진행하고, 나중에 서비스를 끊고 작업을 진행하는 정기적인 PM 작업시에 정상적인 마이그레이션 작업을 진행할 수 있다.

```
----- DAUM Web Server -----          ----- Web Client -----
    httpd(80)          <----------------   http://www.daum.net
    /was/index.html
```

작업: /was ---> /zeus
# ln -s /was /zeus

(실무 예) 관리 디렉토리 통합
- 하나의 서버(물리적인 머신)에 여러가지 서비스가 올라가 있는 경우에, 각 각의 서비스를 관리를 편하게 하기 위해서 관리용 디렉토리 생성하고 그 디렉토리에 모든 서비스 관리 폴더를 모아서 관리하면 좋을 것이다. 이때, 심볼릭 링크가 사용될 수 있다.

```
(필요하면 명령어 수행) (on server1)
# yum install vsftpd ftp       /* FTP 서버/클라이언트 패키지 */
# yum install bind bind-utils  /* DNS 서버 패키지 */
# yum install httpd mod_ssl    /* HTTP 서버 패키지 */

FTP : /etc/vsftpd
DNS : /var/named
WEB : /var/www/html

# cd /test ; rm -rf /test/*
# ln -s /etc/vsftpd FTP       /* FTP 서비스 프로그램 디렉토리 */
# ln -s /var/named DNS        /* DNS 서비스 프로그램 디렉토리 */
# ln -s /var/www/html WEB     /* WEB 서비스 프로그램 디렉토리 */
# cd /test ; ls
# cd FTP
# cd ../DNS
# cd ../WEB
```

(실무 예) 버전 관리
- 소스 코드의 버전 관리를 위해서 심볼릭 링크를 사용할 수 있다. 이전 코드의 설정 파일을 새로운 버전의 설정 파일로 그대로 가지고 올수 있도록 하기위해서 디렉토리 명칭을 일관되게 유지해야 하는 경우에 심볼릭 링크가 사용될 수 있다.

```
/usr/local/tomcat-4.X ----> /usr/local/tomcat
/usr/local/tomcat-5.X ----> /usr/local/tomcat
/usr/local/tomcat-6.X ----> /usr/local/tomcat

(초기 프로그램을 설치하는 경우)
# cd /usr/local
# ls
tomcat-4.X

# ln -s tomcat-4.x tomcat
# cd tomcat
설정 작업(EX: /usr/local/tomcat)
```

(버전 업그레이드 하는 경우)
```
# cd /usr/local
# ls
tomcat-4.X  tomcat-5.X  tomcat

# rm tomcat
# ln -s tomcat-5.x tomcat

# cd tomcat
```
이전설정(/usr/local/tomcat) 복사

**(실무 예)** 하드링크에 대한 이해
- 하드 링크는 파일의 데이터 블럭을 하나만 사용하기 때문에 용량은 같은 용량이라고 표시가 되어도 하나의 공간만 사용되는 것이다.

■ /home 파일 시스템 남은 용량 확인
```
# df -h /home
```

■ /home/file1 500M 파일 생성
```
# cd /home
# dd if=/dev/zero of=/home/file1 bs=1M count=500
```

■ /home 파일 시스템 남은 용량 확인
```
# df -h /home
```

■ 하드링크 생성
```
# ln file1 file2
# ln file1 file3
# ln file1 file4
# ls -l
```

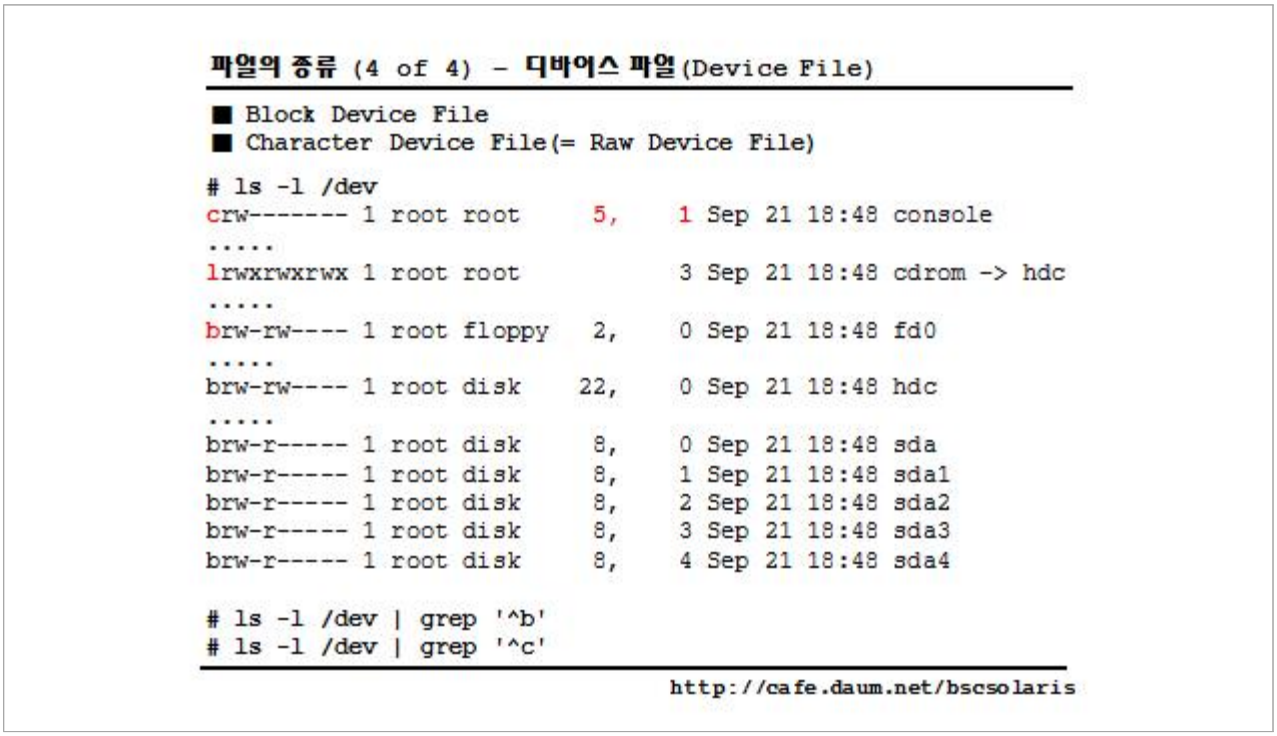
■ /home 파일 시스템 남은 용량 확인
```
# df -h /home
```

■ 파일 삭제 실습
```
# rm -f file4
# df -h /home

# rm -f file1
# df -h /home

# rm -f file2 file3
# df -h /home
```
[사용] deduplecation(중복 제거)

[참고] find /home -inum 139 -type f
[참고] find /home -inum 139 -type f -exec rm -f {} W;




## 4 디바이스 파일 (Device File)

■ 용어(Terms)
- 장치(Device)
- 장치 드라이버(Devie Driver)
- 운영체제(커널)
- 장치 파일(Device File, Special File, Special Device File)
  - /dev/* (예) /dev/pts/2, /dev/sda
  - (WIN) 장치관리자(devmgmt.msc)

■ 디바이스 파일(장치 파일, Device File)
- 블럭 디바이스 파일(Block Device File)
- 캐릭터 디바이스 파일(Character Device File) = Raw Device File

■ 블럭 디바이스 파일
- 블럭 단위로 I/O 발생
- 디스크 디바이스(Disk Device)이면, I/O 단위는 블록 단위(4K(4096 Bytes))

■ 캐릭터 디바이스 파일
- 바이트 단위로 I/O 발생
- 디스크 디바이스(Disk Device)이면 I/O 단위는 512 Bytes(1 Sector = 512 bytes)

■ Major Device 번호
- 장치의 종류
- 장치의 종류가 틀리면 Major Device 번호가 틀리다.

■ Minor Device 번호
- 개별적인 장치의 종류 또는 동작 방법의 차이
- 개별적인 장치의 종류가 틀리면 Minor Device 번호가 틀리다.
- 또는 같은 장치라도 동작 방법이 틀리면 Minor Device 번호가 틀리다.

[EX1] 블럭 디바이스/캐릭터 디바이스 파일 확인

```
[root@server1 ~]# ls -l /dev | grep '^b'
brw-rw----  1 root disk    253,   0 Feb 17 09:45 dm-0
brw-rw----  1 root disk    253,   1 Feb 17 09:45 dm-1
brw-rw----  1 root disk    253,   2 Feb 17 09:45 dm-2
brw-rw----  1 root disk      8,   0 Feb 17 09:45 sda
brw-rw----  1 root disk      8,   1 Feb 17 09:45 sda1
brw-rw----  1 root disk      8,   2 Feb 17 09:45 sda2
brw-rw----+ 1 root cdrom    11,   0 Feb 17 09:45 sr0
```

```
[root@server1 ~]# ls -l /dev | grep '^c'
crw-r--r--  1 root root     10, 235 Feb 17 09:45 autofs
crw-------  1 root root      5,   1 Feb 17 09:45 console
crw-------  1 root root     10,  62 Feb 17 09:45 cpu_dma_latency
crw-rw----  1 root video    29,   0 Feb 17 09:45 fb0
crw-rw-rw-  1 root root      1,   7 Feb 17 09:45 full
crw-rw-rw-  1 root root     10, 229 Feb 17 09:59 fuse
crw-------  1 root root    244,   0 Feb 17 09:45 hidraw0
crw-------  1 root root     10, 228 Feb 17 09:45 hpet
..... (중략) .....
```

[실습] 하드웨어 목록 확인 하기

■ 실습 시스템
- server1

■ 실습 시나리오
- HW 장치 목록 확인 하기

[EX] HW 장치 목록 확인 하기 - HW 장치 확인 명령어

① (server1) CPU 아키텍쳐 정보 확인

```
[root@server1 ~]# ls[TAB][TAB]
ls        lscpu     lsiio     lslocks   lsmd      lsns      lsscsi
lsattr    lsgpio    lsinitrd  lslogins  lsmem     lsof      lsusb
lsblk     lshw      lsipc     lsmcli    lsmod     lspci     lsusb.py
```

```
[root@server1 ~]# lscpu
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              4
On-line CPU(s) list: 0-3
Thread(s) per core:  1
Core(s) per socket:  4
Socket(s):           1
NUMA node(s):        1
Vendor ID:           GenuineIntel
CPU family:          6
Model:               42
Model name:          Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz
Stepping:            7
CPU MHz:             3401.000
BogoMIPS:            6802.00
Hypervisor vendor:   VMware
Virtualization type: full
L1d cache:           32K
L1i cache:           32K
L2 cache:            256K
L3 cache:            6144K
NUMA node0 CPU(s):   0-3
Flags:               fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush
mmx fxsr sse sse2 ss ht syscall nx rdtscp lm constant_tsc arch_perfmon nopl xtopology tsc_reliable
nonstop_tsc cpuid pni pclmulqdq ssse3 cx16 pcid sse4_1 sse4_2 x2apic popcnt tsc_deadline_timer aes
xsave avx hypervisor lahf_lm pti ssbd ibrs ibpb stibp tsc_adjust arat flush_l1d arch_capabilities
```

[root@server1 ~]# lscpu -e

```
CPU NODE SOCKET CORE L1d:L1i:L2:L3 ONLINE
0   0    0      0    0:0:0:0       yes
1   0    0      1    1:1:1:0       yes
2   0    0      2    2:2:2:0       yes
3   0    0      3    3:3:3:0       yes
```

- -e, --extended[=<list>] print out an extended readable format

② (server1) MEMEORY 정보 확인

[root@server1 ~]# lsmem

```
RANGE                                  SIZE  STATE REMOVABLE BLOCK
0x0000000000000000-0x000000000fffffff  256M online       no   0-1
0x0000000010000000-0x0000000017ffffff  128M online      yes     2
0x0000000018000000-0x00000000bfffffff  2.6G online       no  3-23

Memory block size:       128M
Total online memory:       3G
Total offline memory:      0B
```

- free CMD
- top CMD

③ (server1) SCSI device 정보 확인

[root@server1 ~]# lsscsi

```
[1:0:0:0]    cd/dvd  NECVMWar VMware IDE CDR10 1.00  /dev/sr0
[2:0:0:0]    disk    VMware,  VMware Virtual S 1.0   /dev/sda
```

④ (server1) Block Device 정보 확인

[root@server1 ~]# lsblk

```
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda           8:0    0   60G  0 disk
 ├─sda1        8:1    0    1G  0 part /boot
 └─sda2        8:2    0   59G  0 part
   ├─cl-root 253:0    0 37.6G  0 lvm  /
   ├─cl-swap 253:1    0    3G  0 lvm  [SWAP]
   └─cl-home 253:2    0 18.4G  0 lvm  /home
sr0          11:0    1  7.7G  0 rom  /run/media/root/CentOS-8-2-2004-x86_64-dvd
```

- fdisk/gdisk CMD
- parted CMD

⑤ (server1) PCI 정보 확인

[root@server1 ~]# lspci

```
.... (중략) ...
00:00.0 Host bridge: Intel Corporation 440BX/ZX/DX - 82443BX/ZX/DX Host bridge (rev 01)
00:01.0 PCI bridge: Intel Corporation 440BX/ZX/DX - 82443BX/ZX/DX AGP bridge (rev 01)
00:07.0 ISA bridge: Intel Corporation 82371AB/EB/MB PIIX4 ISA (rev 08)
00:07.1 IDE interface: Intel Corporation 82371AB/EB/MB PIIX4 IDE (rev 01)
00:07.3 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 08)
00:07.7 System peripheral: VMware Virtual Machine Communication Interface (rev 10)
00:0f.0 VGA compatible controller: VMware SVGA II Adapter
00:10.0 SCSI storage controller: Broadcom / LSI 53c1030 PCI-X Fusion-MPT Dual Ultra320 SCSI (rev 01)
00:11.0 PCI bridge: VMware PCI bridge (rev 02)
00:15.0 PCI bridge: VMware PCI Express Root Port (rev 01)
00:15.1 PCI bridge: VMware PCI Express Root Port (rev 01)
..... (중략) .....
00:18.6 PCI bridge: VMware PCI Express Root Port (rev 01)
00:18.7 PCI bridge: VMware PCI Express Root Port (rev 01)
02:00.0 USB controller: VMware USB1.1 UHCI Controller
02:01.0 Ethernet controller: Intel Corporation 82545EM Gigabit Ethernet Controller (Copper) (rev 01)
02:02.0 USB controller: VMware USB2 EHCI Controller
```

⑥ (server1) USB 장치 확인

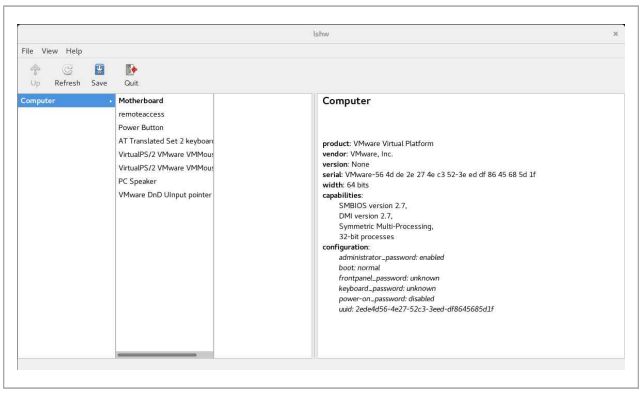
[root@server1 ~]# lsusb

```
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 003: ID 0e0f:0002 VMware, Inc. Virtual USB Hub
Bus 002 Device 002: ID 0e0f:0003 VMware, Inc. Virtual Mouse
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
```

⑦ (server1) 전체 하드웨어에 대한 정보
[참고] 필요하면 명령어 수행
**# yum install lshw lshw-gui**

(CLI) # lshw
(GUI) # lshw-gui &

**[root@server1 ~]# lshw-gui**



[참고] 여러 가지 종류의 파일 생성 및 삭제 방법

■ 일반 파일(Regular File) 생성

```
# echo 1111 > file1.txt
```

■ 디렉토리 파일(Directory File) 생성

```
# mkdir dir1
```

■ 링크 파일(Link File) 생성

```
(심볼릭 링크) # ln -s file1 file1.link
(하드 링크)   # ln file1 file1.link
```

■ 장치 파일(Device File) 생성

* 직접 생성은 가능하지만 실제 장치 드라이버와 연동되지 않으면 의미 없음.)

```
(블록장치) # ls -l /dev/sda
(문자장치) # ls -l /dev/tty
# mknode myblockfile b 8 0
# mknode mycharfile c 4 64
```

■ 소켓 파일(Socket File) 생성

```
(소켓 파일 찾기)
# find / -name '*.sock' -type s
(소켓 파일 생성 예시 - 간단한 예)
# yum -y install socat
# socketfile=/tmp/mysocket && rm -f /tmp/$socketfile
# socat -d -d UNIX-LISTEN:$socketfile,fork EXEC:/bin/cat &
# ls -l $socketfile
# file $socketfile
```

■ 파이프 파일

```
(파이프 파일 생성)
# mkfifo mypipe
# ls -l mypipe
# file mypipe
(파이프 파일 테스트)
[TERM1] # cat < mypipe
[TERM2] # echo "HELLO PIPE" > mypipe
```

■ 파일 삭제

```
# rm -f (파일이름)
```

다음 스크립트를 사용하여 /test 디렉토리에 테스트용 파일 종류을 생성할 수도 있다.

```
mkdir -p /test && cd /test && rm -rf /test/*
# 일반 파일 생성
echo "Hello World" > myregular
# 디렉토리 파일 생성
mkdir -p mydirectory
# 링크 파일 생성
ln -sf myregular hardlink
ln -f myregular symlink
# 장치 파일
sudo mknod mkblock b 8 0
sudo mknod mychar c 4 64
# 소켓 파일
dnf -q -y install socat
rm -f /test/mysocket
socat -d -d UNIX-LISTEN:/test/mysocket,fork EXEC:/bin/cat &
# 파이프 파일
mkfifo /test/mypipo
# 확인
ls -l /test/
```

```
# 파일 삭제
rm -rf /test/*
```