부록 III: Unix 명령어 요약

Unix 명령어 요약 목차

1. 파일과 디렉토리 다루기

컴퓨터를 사용하면서 아마도 가장 많이 하는 일은 파일과 디렉토리를 다루는 작업일 것이다. Unix나 Linux에서도 역시 파일과 디렉토리를 다루는 것은 상당히 중요하다. 윈도우에서는 탐색기라는 하나의 프로그램이 파일 목록을 보여주거나 복사, 이동 등의 모든 작업을 다 떠맡고 있지만 Unix나 Linux에서는 각각의 작업마다 그 한 가지 일만을 수행하는 프로그램이 따로 있다. 이제 파일과 디렉토리를 다루는 명령어에 대해서 익혀보자.

1-1. 로그인과 로그아웃 logout, exit

우선 리눅스로 부팅한 후 로그인해보자. 로그인 프롬프트가 나타나면 계정과 패스워드를 입력하면 로그인할 수 있다.

```
linux login : namul
Password :

[namul@dcclab namul]$ _
[namul@dcclab namul]$ logout
```

'[namul@dcclab namul]'를 보통 쉘 프롬프트 또는 커맨트 프롬프트라고 한다. 커서가 깜박이는 곳이 명령어를 입력하는 곳이다.

로그인을 했는데 다시 앞의 화면으로 돌아가는 방법을 모르니 막막하다. 윈도우와 달리 시스템 종료는 루트 계정으로만 할 수 있다. 그렇다고 그냥 전원을 내리는 것도 불안하고. 로그아웃을 할 때는 프롬프트에서 다음과 같이 logout이나 exit를 입력하면 된다.

1-2. 파일 목록 보기 ls

디렉토리에 있는 파일 목록을 나열하는 명령어는 ls이다. ls의 사용법은 다음과 같다.

```
ls [옵션] [파일]
```

[옵션]과 [파일] 모두 생략될 수 있으며, [파일]자리에는 파일이나 디렉토리가 들어갈 수 있다. GNU/Linux는 디렉토리 역시 하나의 특수한 파일로 취급한다. [파일]이 생략되면 현재 작업중인 디렉토리에 있는 파일의 목록이 나열된다.

```
[namul@dcclab namul]$ ls
[namul@dcclab namul]$ _
```

아무것도 나오지 않는 것은 현재 디렉토리에 파일이 없기 때문이다. 루트 디렉토리에 있는 파일을 나열하면 다음과 같다.

```
[namul@dcclab namul]$ ls /
bin   data  etc   initrd  lost+found  mnt   proc  sbin      tmp  var
boot  dev   home  lib     misc        opt   root  tftpboot  usr
[namul@dcclab namul]$
```

Unix나 Linux의 대부분의 명령어에는 옵션이 있다. 옵션은 보통 - 뒤에 영문자나 숫자 한 글자 혹은 -- 뒤에 영어 단어가 붙어있는 형식으로 구성된다. 다음과 같이 입력해보자.

```
[namul@dcclab namul]$ ls -l /
total 206
drwxr-xr-x    2 root     root         4096 Apr 11  2003 bin
drwxr-xr-x    4 root     root         2048 Apr 11  2003 boot
drwxr-xr-x    6 root     root         4096 Nov 10 14:56 data
drwxr-xr-x   20 root     root       118784 Oct 10 19:43 dev
drwxr-xr-x   81 root     root         8192 Jan 19 15:29 etc
drwxr-xr-x   23 root     root         4096 Jan  8 11:31 home
drwxr-xr-x    2 root     root         4096 Jun 22  2001 initrd
drwxr-xr-x    8 root     root         4096 Apr 15  2003 lib
drwx------    2 root     root        16384 Mar  5  2003 lost+found
drwxr-xr-x    2 root     root         4096 Aug 27  2002 misc
drwxr-xr-x    4 root     root         4096 Jun 24  2002 mnt
drwxr-xr-x    2 root     root         4096 Aug 24  1999 opt
dr-xr-xr-x   79 root     root            0 Oct 11 04:42 proc
drwxr-x---   13 root     root         4096 Jan 31 00:46 root
drwxr-xr-x    2 root     root         8192 Apr 15  2003 sbin
drwxr-xr-x    3 root     root         4096 Mar  5  2003 tftpboot
drwxrwxrwt    6 root     root         4096 Feb  1 15:46 tmp
drwxr-xr-x   17 root     root         4096 Dec 16 19:12 usr
drwxr-xr-x   29 root     root         4096 Jul 30  2003 var
lrwxr-xr-x    1 root     root           19 Jul 24  2003 test -> /home/namul/test
[namul@dcclab namul]$ _
```

-l은 보다 자세한 정보를 보여주는 옵션이다. 앞의 화면을 보면 파일에 대한 여러 가지 정보를 알 수 있다. 한 줄을 뽑아서 살펴보자.

```
drwxr-xr-x    2 root     root         4096 Apr 11  2003 bin
```

먼저 맨 앞의 drwxr-xr-x에서 첫 번째 글자는 파일의 종류를 나타낸다. 이 경우처럼 첫 번째 문자가 d일 경우 그 파일은 디렉토리이다. 보통의 파일은 -, 디렉토리는 d, 캐릭터 장치 파일은 c, 블록 장치 파일은 b로 표시한다. l은 해당 파일이 다른 파일에 대한 심볼릭 링크라는 뜻이다. 심볼릭 링크란 윈도우의 바로가기나 단축 아이콘과 비슷하지만 훨씬 유연해서 실제 파일과 똑같은 방식으로 다룰 수 있다.

파일의 종류를 나타내는 첫 글자 뒤의 아홉 글자는 파일에 대한 접근 권한을 나타낸다. 그 중 처음 세 글자는 소유자(owner)의 권한, 그 다음 세 글자는 소유 그룹(group)의 권한, 마지막 세 글자는 나머지 다른 사용자의 권한을 나타내며, r은 읽기, w는 쓰기, x는 실행 권한을 의미한다. 예를 들어 앞에서와 같이 rwxr-x---로 되어 있다면, 파일의 소유자는 그 파일을 읽고(r), 수정하고(w), 실행(x)할 수 있으며, 그룹은 읽거나(r) 실행할(x) 수는 있지만 수정하거나 지울 수는 없고, 그 외의 사람은 그 파일을 읽고 실행하고 수정하는 것이 모두 허용되지 않는다.

파일의 종류와 접근 권한을 나타내는 열 글자 뒤의 숫자는 하드 링크 번호를 의미하며, 그 뒤의 두 단어는 각각 파일의 소유자와 그룹을 나타낸다. 앞의 예에서는 소유자와 그룹 모두 root이다. Linux나 전통적인 Unix에는 파일의 소유자와 그룹이라는 개념이 있어서 사용자마다 파일에 접근할 수 있는 권한이 다르다. 이것은 도스나 윈도우와는 달리 다중 사용자용 운영체제이기 때문이다. 파일에 대한 접근 권한은 파일의 소유자와 슈퍼유저(root)만이 변경할 수 있다.

그룹 뒤의 숫자는 파일의 크기를 나타내며, 그 다음은 파일이 마지막으로 수정된 시간을 나타낸다. 예를 들어 앞의 bin이라는 파일은 크기가 4096바이트 이며 마지막으로 수정된 시간은 2003년 4월 11일이다. 마지막으로 맨 오른쪽에 파일명이 표시되며, 그 파일이 심볼릭 링크일 경우 어떤 파일에 대한 심볼릭 링크인지 표시된다. 예를 들어 위의 예제 화면 (57p)에서 test라는 파일은 /home/namul 디렉토리에 있는 test라는 파일에 대한 심볼릭 링크이다.

ls에서 -l과 함께 사용되는 옵션으로 -a가 있다.

```
[namul@dcclab namul]$ ls -a
.   .bash_history   .bash_profile   .canna   .gtkrc   .viminfo
..  .bash_logout    .bashrc         .emacs   .kde
[namul@dcclab namul]$ ls -al
total 52
drwxr-xr-x    4 namul    namul        4096 Nov 13 23:08 .
drwxr-xr-x   23 root     root         4096 Jan  8 11:31 ..
-rw-------    1 namul    namul        221 Nov 14 01:27 .bash_history
-rw-r--r--    1 namul    namul         24 Nov 13 23:05 .bash_logout
-rw-r--r--    1 namul    namul        191 Nov 13 23:05 .bash_profile
-rw-r--r--    1 namul    namul        124 Nov 13 23:05 .bashrc
-rw-r--r--    1 namul    namul       5531 Nov 13 23:05 .canna
-rw-r--r--    1 namul    namul        854 Nov 13 23:05 .emacs
-rw-r--r--    1 namul    namul        120 Nov 13 23:05 .gtkrc
drwxr-xr-x    3 namul    namul       4096 Nov 13 23:05 .kde
-rw-------    1 namul    namul        638 Nov 13 23:08 .viminfo
drwxrwxr-x    2 namul    namul       4096 Nov 13 23:08 public_html
[namul@dcclab namul]$
```

-a는 숨은 파일까지 모두 나열하라는 옵션이다. Linux에서 숨은 파일은 .으로 시작한다. 다시 말해서 .으로 시작하는 파일은 모두 숨은 파일로 간주한다. 앞과 같이 ls에서 -a 옵션을 붙여 사용하면 .으로 시작하는 파일까지 모두 나열된다.

```
[namul@dcclab namul]$ ls -al b*      <- b로 시작하는 모든 파일을 찾아서 나열
[namul@dcclab namul]$ ls -al .*his*  <- .으로 시작하고(즉, 숨은 파일) 중간에 his가 들어
있는 모든 파일을 나열한다.
```

위에서와 같이 정규표현 식을 이용하여 특정 파일을 찾아볼 수 있다.

1-3. 디렉토리 이동과 생성 및 삭제 cd, mkdir, rmdir

이제 다른 디렉토리로 이동해보자. 다른 디렉토리로 이동하는 명령어는 cd이다. 사용법은 다음과 같다.

```
cd [디렉토리]
```

다음과 같이 입력해보자.

```
[namul@dcclab namul]$ pwd
/home/namul
[namul@dcclab namul]$ cd /usr
[namul@dcclab namul]$ pwd
/usr
[namul@dcclab namul]$ cd ~
[namul@dcclab namul]$ pwd
/home/namul
```

pwd는 현재 작업중인 디렉토리를 표시하는 명령어이다. cd /usr은 루트 디렉토리에 있는 usr이라는 디렉토리로 이동하라는 뜻이다. Linux에서 ~는 루트 디렉토리를 의미하는 /와 함께 특별한 의미를 갖는 것으로서, 사용자의 홈 디렉토리를 의미한다. 보통 사용자의 홈 디렉토리는 /home 디렉토리 안에 들어 있다. 예를 들어서 namul이라는 사용자의 홈 디렉토리는 /home/namul이 된다. root를 제외한 나머지 사용자는 자신의 홈 디렉토리 밖에 있는 파일에 대해서는 읽기와 실행 권한만 갖는다.

이번에는 디렉토리를 만들어보자. 디렉토리를 만드는 명령어는 mkdir이다.

```
mkdir [옵션] [디렉토리]
```

예를 들어 test라는 디렉토리를 만들고 싶다면 다음과 같이 입력한다.

```
[namul@dcclab namul]$ mkdir test
[namul@dcclab namul]$ ls
test
[namul@dcclab namul]$ cd test
```

mkdir에서 자주 사용되는 옵션은 -p이다. -p 옵션을 사용하면 필요한 경우 하위 디렉토리까지 만들게 된다. 예를 들어, mkdir -p first/second/third와 같이 입력하면 현재 디렉토리 안에 first라는 디렉토리가 만들어지고, 또 그 안에 second, 또 그 안에 third라는 디렉토리가 만들어진다.

디렉토리를 지우는 명령어는 rmdir이다. 이때 대상 디렉토리 안에는 파일이 하나도 없어야 한다. 사용법은 다음과 같다.

```
rmdir [옵션] [디렉토리]
```

앞에서 만든 test라는 디렉토리를 지워보자.

```
[namul@dcclab test]$ cd
[namul@dcclab namul]$ rmdir test
[namul@dcclab namul]$ ls
[namul@dcclab namul]$
```

1-4. 파일의 복사, 이동, 삭제, 링크 cp, mv, rm, ln

Linux에서 파일을 복사할 때에는 cp라는 명령어를 이용한다. 사용법은 다음과 같다.

```
cp [옵션] [원본파일] [대상파일]
```

다음과 같이 입력해보자.

```
[namul@dcclab namul]$ ls
[namul@dcclab namul]$ cp /etc/services ./
[namul@dcclab namul]$ ls
services
[namul@dcclab namul]$ _
```

/etc 디렉토리에 있는 services라는 파일을 현재 디렉토리로 복사해온 것이다. 복사하면서 파일의 이름을 변경할 때에는 다음과 같이 입력하면 된다.

```
[namul@dcclab namul]$ ls
services
[namul@dcclab namul]$ cp services services2
[namul@dcclab namul]$ ls
services services2
[namul@dcclab namul]$ _
```

잠깐! 여기서 짚고 넘어가야 할 것이 있다. 만약 services2라는 디렉토리가 존재한다면 앞의 명령어는 services라는 파일을 services2라는 이름으로 복사를 하는 것이 아니라 services라는 디렉토리에 services라는 이름으로 복사된다.

cp에는 많은 옵션이 있는데 자주 사용되는 옵션을 정리하면 다음과 같다.

--help	도움말을 보여준다.
-d, --no-deference	만약 복사할 원본이 심볼릭 링크 파일이면 링크 대상이 되는 파일을 복사하는 대신 그 심볼릭 링크 파일 자체를 심볼릭 정보와 함께 복사한다.
-f, --force	만약 복사하려고 하는 이름과 같은 이름의 읽기 전용 파일이 이미 있어도 그 파일을 강제로 지우고 복사한다.
-i, --interactive	만약 복사하려고 하는 이름과 같은 이름의 파일이 이미 있으면 사용자에게 어떻게 처리할 것인지 물어본다.
-p, --preserve	원본 파일의 소유자, 그룹, 권한, 시간 정보를 그대로 보존하여 복사한다.
-r	일반 파일이면 그냥 복사하고, 원본이 디렉토리이면 그 디렉토리 안에 있는 파일이나 모든 서브 디렉토리 및 그 안에 있는 파일까지 모두 복사한다.
-R, --recursive	-r 옵션과 같지만 복사하려는 파일이 특별 장치 파일일 경우 속성을 그대로 유지하여 복사한다. 즉 /dev 디렉토리 안의 파일을 복사할 경우 -r 대신 -R을 사용해야 한다.
-a, --archive	-dpR과 같다.
-v, --verbose	파일을 복사하는 과정을 상세히 보여준다.

파일이나 디렉토리를 이동하거나 이름을 변경하는 데 사용하는 명령어는 mv이다. mv의 사용법은 cp와 거의 비슷하다.

```
mv [옵션] [원본파일] [대상파일]
```

```
[namul@dcclab namul]$ ls
services services2
[namul@dcclab namul]$ mv services2 services3
[namul@dcclab namul]$ ls
services services3
[namul@dcclab namul]$ _
```

앞에서는 services2라는 파일을 services3라는 파일로 이름을 변경한 것이다. 만약 이미 services3라는 디렉토리가 존재한다면 앞의 명령은 services2라는 파일을 services3라는 디렉토리 안으로 이동시키는 명령이 된다. mv에서 자주 사용되는 옵션을 정리하면 다음과 같다.

--help	도움말을 보여준다.
-f, --force	만약 복사하려고 하는 이름과 같은 이름의 읽기 전용 파일이 이미 있어도 그 파일을 강제로 지우고 복사한다.
-i, --interactive	만약 복사하려고 하는 이름과 같은 이름의 파일이 이미 있으면 사용자에게 어떻게 처리할 것인지 물어본다.
-v, --verbose	파일을 옮기는 과정을 상세히 보여준다.

cp와 mv 명령에서 주의해야 할 점은 대상 파일이 이미 있으면 묻지 않고 그냥 덮어 써 버린다는 점이다. 따라서 명령을 사용할 때에는 항상 미리 대상 파일이 존재하는지 주의해서 사용해야 한다. 어떤 사람은 항상 -i 옵션을 사용하도록 alias를 지정해 사용할 것을 추천하기도 한다. mv나 cp에서 항상 -i 옵션을 사용하려면 다음과 같이 alias를 지정해주면 된다.

```
[namul@dcclab namul]$ alias cp='cp -i'
[namul@dcclab namul]$ alias mv='mv -i'
[namul@dcclab namul]$ alias
alias cp='cp -i'
alias mv='mv -i'
[namul@dcclab namul]$ _
```

이제 복사한 파일을 지워보자. 이때 사용하는 명령어는 rm이다.

```
rm [옵션] [파일]
```

앞에서 mv 명령으로 이름을 변경한 services3 파일을 삭제해보자.

```
[namul@dcclab namul]$ ls
services services3
[namul@dcclab namul]$ rm services3
[namul@dcclab namul]$ ls
services
[namul@dcclab namul]$ _
```

rm에서 자주 사용되는 옵션을 정리하면 다음과 같다.

--help	도움말을 보여준다.
-f, --force	지우려는 파일이 없을 경우 아무런 에러 메시지도 표시하지 않는다. 또 지우려는 파일이 읽기전용 파일이라도 지운다.
-i, --interactive	각 파일을 하나씩 지울 것인지 사용자에게 일일이 물어본다.
-r, -R, --recursive	일반 파일이면 그냥 지우고, 디렉토리이면 그 하위 디렉토리와 그 안의 파일까지 모두 지운다.

만약 디렉토리 안에 파일이 있을 때 rmdir을 이용하여 디렉토리를 삭제하려면 그 안에 있는 파일을 모두 지운 후 rmdir 명령으로 디렉토리를 삭제해야 하지만, rm -r 명령을 이용하면 한번에 끝낼 수 있다. 그러나 편리한 만큼 위험한 명령어이다. 예를 들어, 루트 권한으로 루트 디렉토리에서 rm -rf *라는 명령을 내린다면 시스템의 모든 파일이 삭제되기 때문이다.

Linux 파일 시스템에는 링크라는 것이 있는데, 링크에는 '심볼릭 링크'와 '하드 링크' 이렇게 두 종류가 있다. 심볼릭 링크는 앞에서 설명했다. 여러 명의 사용자가 한 파일을 공유해야 할 경우가 있다고 생각해보자. 예를 들어 그 파일이 주소록이고, 각각의 사용자가 때때로 그 주소록을 업데이트해야 한다고 생각해보자. 그럴 경우 각각 따로 파일을 만들어 주기적으로 짜 맞추는 것보다는 하나의 중심적인 파일을 만들어두고 각자 그 파일에 연결된 파일을 만들어두면 편리하다. 이런 경우 사용하는 것이 하드 링크이다. 사용법은 다음과 같다.

```
ln [옵션] [원본파일] [대상파일]
```

[대상파일]을 생략하면 원본과 같은 이름으로 현재 작업중인 디렉토리에 링크를 만든다.

```
[namul@dcclab namul]$ ls
services
[namul@dcclab namul]$ ln services services2
[namul@dcclab namul]$ ls
services services2
[namul@dcclab namul]$ _
```

마치 파일이 복사된 것처럼 보인다. 그러나 다음과 같이 한 파일의 내용을 변경하면 다른 파일의 내용도 함께 변경된다. 실제로는 하나의 파일이다. 하드 링크 대신 심볼릭 링크를 만들 때에는 -s 옵션을 사용한다.

```
[namul@dcclab namul]$ ln -s services services3
[namul@dcclab namul]$ ls -l
total 32
-rw-r--r--    2 namul    namul       14450 Feb  2 03:05 services
-rw-r--r--    2 namul    namul       14450 Feb  2 03:25 services2
lrwxrwxrwx    1 namul    namul           8 Feb  2 03:40 services3 -> services
[namul@dcclab namul]$ _
```

ln에서 자주 사용되는 옵션은 다음과 같다.

-d, -F, --directory	디렉토리의 하드 링크를 허용한다. 이것은 시스템 관리자(root)만이 할 수 있다.
-f, --force	이미 링크하려는 이름과 같은 이름의 파일이 있다고 해도 그 파일을 강제로 지우고 링크한다.
-i, --interactive	링크하려는 이름과 같은 이름의 파일이 이미 있으면 사용자에게 어떻게 처리할 것인지를 물어본다.
-n, --no-deference	링크할 원본 파일이 심볼릭 링크이면 그 링크의 대상 파일에 대한 심볼릭 링크를 만든다.
-s, --symbolic	하드 링크 대신 심볼릭 링크를 만든다.

1-5. 파일의 접근 권한 변경 chmod, chown, chgrp, umask

파일의 접근 권한을 변경할 때는 chmod를 이용한다. 사용 방법은 다음과 같다.

```
chmod [옵션] [모드] [파일]
```

[모드]를 적는 방법은 두 가지가 있는데 그 첫 번째 방법은 다음과 같다.

```
[ugoa] [+-=] [rwxXstugo]
```

처음에 나오는 [ugoa]는 u, g, o, a 중의 한 글자를 의미하는데 각각 소유자(User), 그룹(Group), 나머지(Others), 모두(All)를 뜻하며, 생략될 경우 모두(a)와 같다. 다음의 [+-=]은 +, -, = 중의 한 글자를 의미하며, +는 권한 부여, -는 권한 박탈, =는 기존 권한 유지를 뜻한다.

맨 끝의 [rwxXstugo] 자리 역시 영어 문자 한 개가 들어가며 r은 읽기, w는 쓰기, x는 실행, X는 이미 어떤 사용자(소유자, 그룹, 나머지 중의 하나 이상)에게 실행 권한이 있을 때만 실행 권한 부여, s는 소유자와 그룹에게만 실행 권한 부여, u는 소유자 권한으로 실행(Set Uid), g는 그룹 권한으로 실행(Set Gid), t는 스티키 비트(Sticky Bit)를 의미한다.

set-uid와 set-gid는 접근 권한의 특별한 경우로 set-uid가 설정되어 있는 파일은 파일을 실행할 때 누가 실행하든 파일의 소유자 권한 실행된다. 예를 들어 PPP 프로토콜로 인터넷에 연결할 때 사용하는 /usr/sbin/pppd 파일은 루트 권한으로 실행해야 한다. 따라서 일반 사용자도 PPP를 사용할 수 있도록 하기 위해서 대부분의 배포본에서 /usr/sbin/ppd 파일은 소유자가 루트이고, set-uid 설정이 되어 있다. set-gid 역시 set-uid와 비슷하지만 소유자의 권한이 아닌 그룹 권한으로 실행된다는 점이 다르다.

set-uid와 set-gid는 예전부터 보안의 구멍이 되어왔다. 어떤 프로그램이 루트 권한으로 실행되는 것은 항상 잠재적인 보안상의 위험을 안고 있는 것이다. 따라서 꼭 필요한 경우가 아니라면 set-uid와 set-gid는 사용하지 않는 것이 좋다. set-uid가 설정되어 있는 경우 ls -l 명령어를 이용해 파일을 열람해보면 소유자의 실행 권한이 x가 아닌 s로 표시되며, set-gid가 설정되어 있으면 그룹의 실행 권한이 s로 표시된다.

```
[namul@dcclab namul]$ ls -l /usr/sbin/pppd
-rwsr-xr-x    1 root     dip        213202 Jul 22  2002 /usr/sbin/pppd
[namul@dcclab namul]$
```

스티키 비트 역시 접근 권한의 특별한 경우로, 이것이 설정되어 잇는 디렉토리 안에 있는 파일은 접근 권한을 어떻게 지정하더라도 소유자와 루트 외에는 접근할 수 없다. 스티키 비트가 설정되어 있는 대표적인 디렉토리는 /tmp이다. 이 디렉토리는 임시 파일이 위치하는 곳이기 때문에 어떤 사용자든 읽고 쓸 수 있어야 한다. 그러나 그냥 아무나 변경할 수 있도록 한다면 다른 사용자가 사용하는 프로그램에서 만든 임시 파일도 모두 변경할 수 있게 될 것이다. 스티키 비트가 지정된 디렉토리는 ls -l 명령어로 열람했을 때 사용자의 실행권한이 x 대신 t로 표시된다.

```
[namul@dcclab namul]$ ln -s services services3
[namul@dcclab namul]$ ls -l
total 32
-rw-r--r--    2 namul    namul       14450 Feb  2 03:05 services
-rw-r--r--    2 namul    namul       14450 Feb  2 03:25 services2
lrwxrwxrwx    1 namul    namul           9 Feb  2 03:40 services3 -> services2
[namul@dcclab namul]$ chmod g+w services
[namul@dcclab namul]$ ls -l
total 32
-rw-rw-r--    2 namul    namul       14450 Feb  2 03:05 services
-rw-rw-r--    2 namul    namul       14450 Feb  2 03:25 services2
lrwxrwxrwx    1 namul    namul           9 Feb  2 03:40 services3 -> services2
[namul@dcclab namul]$ _
```

chmod 명령을 이용하여 그룹(Group)에 쓰기(w) 권한을 부여한 후 ls -l 명령으로 확인해 본 모습이다. 앗, 그런데 services 파일의 권한을 변경했는데 services2의 권한도 변경되었다! chmod의 버그일까? 그렇지 않다. 앞에서 services2 파일을 어떻게 만들었는지 생각해보라.

services2 파일은 ln 명령을 사용하여 만들었다. 즉, services2 파일은 services 파일의 링크이므로 실제로는 같은 파일이다. 따라서 한 파일의 권한을 변경하면 다른 파일의 권한도 변경되는 것이다.

[모드]를 적는 두 번째 방법은 숫자를 이용하는 방법이다. 읽기 권한은 4, 쓰기 권한은 2, 실행 권한은 1로 계산하여 부여하고 싶은 권한을 모두 합하여 적는다. 예를 들어 읽기, 쓰기, 실행 모두 가능은 4+2+1=7이고, 읽기와 실행만 가능은 4+1=5이고, 0은 아무런 권한도 없음을 의미한다. 이렇게 만들어진 숫자를 소유자, 그룹, 나머지 순으로 적는 것이다. 예를 한번 들어보자.

```
[namul@dcclab namul]$ ln -s services services3
[namul@dcclab namul]$ ls -l
total 32
-rw-r--r--    2 namul    namul       14450 Feb  2 03:05 services
-rw-r--r--    2 namul    namul       14450 Feb  2 03:25 services2
lrwxrwxrwx    1 namul    namul           9 Feb  2 03:40 services3 -> services2
[namul@dcclab namul]$ chmod 600 services
[namul@dcclab namul]$ ls -l
total 32
-rw-------    2 namul    namul       14450 Feb  2 03:05 services
-rw-------    2 namul    namul       14450 Feb  2 03:25 services2
lrwxrwxrwx    1 namul    namul           9 Feb  2 03:40 services3 -> services2
[namul@dcclab namul]$ _
```

파일의 소유자에게만 읽기(4)와 쓰기(2) 권한을 부여하고 나머지 사용자에게는 아무 권한도 주지 않도록(0) 한 것이다.

앞에서 언급했던 문자로 지정하는 방법과 다른 점은 앞에서 설명한 방법이 권한을 상대적으로 부여하거나 박달하는 데 비해 숫자로 지정하는 방법은 이미 존재하는 권한에 관계없이 권한을 절대적으로 지정한다는 점이다. 따라서 앞의 방법을 상대 지정 방법이라고 하고, 뒤의 방법을 절대 지정 방법이라고 하기도 한다.

파일의 권한을 변경하는 것은 파일의 소유자만이 할 수 있다. root가 아니면 거의 쓸 수 없다. 파일의 소유자를 변경할 때에는 chown 명령어를 사용하고, 그룹을 변경할 때에는 chgrp 명령어를 사용한다.

```
chown [옵션] [소유자] [.그룹] [파일]
chgrp  [옵션] [그룹] [파일]
```

앞에서 작성했던 test 파일의 소유자와 그룹을 변경해보자.

```
[namul@dcclab namul]$ su
Password :
[root@dcclab namul]$ ls -l test
total 34
-rw-r--r--    2 namul    namul         75 Feb  2 03:05 test
[namul@dcclab namul]$ chown root test
[namul@dcclab namul]$ ls -l test
-rw-r--r--    2 root     namul         75 Feb  2 03:05 test
[namul@dcclab namul]$ chown namul.namul test
[namul@dcclab namul]$ ls -l test
total 34
-rw-r--r--    2 namul    namul         75 Feb  2 03:05 test
[namul@dcclab namul]$ _
```

소유자와 그룹을 한꺼번에 변경하고 싶을 때는 앞에서와 같이 chown [소유자.그룹] [파일] 식으로 적어주면 된다. 예를 들어 sample라는 파일의 소유자와 그룹을 모두 namul로 변경하고 싶다면 다음과 같이 입력한다.

```
chown namul.namul sample
```

지금까지 파일의 권한을 설정하는 방법에 대해 알아보았다. 파일을 생성하게 되면 기본적으로 주어지는 사용 권한을 umask라 한다. 사용법은 다음과 같다.

```
umask [모드]
```

기본 사용 권한은 새 파일이나 디렉토리를 만들 때마다 시스템에 의해 지정되며, 기본 값은 /etc/profile에 존재하며, 최초의 umask 값은 022이다.

```
[namul@dcclab namul]$ umask
22
[namul@dcclab namul]$ cat > test.txt
test
^d
[namul@dcclab namul]$ ls -al
total 12
drwxrwxr-x    2 namul    namul        4096 Feb 19 00:54 .
drwxr-xr-x   11 namul    namul        4096 Feb 19 00:53 ..
-rw-r--r--    1 namul    namul           8 Feb 19 00:53 test.txt
```

022라고 설정되어 있다면, 생성된 파일의 사용권한은 644(-rw-r--r--)로 생성되게 된다. umask는 chmod 명령어와 반대의 개념을 가지고 있다 생각하면 된다.

umask는 일반적으로 /etc/profile에서 설정하여 시스템 전체를 기본 설정으로 하거나 개인적으로 설정을 원할 경우에는 .profile이나 .bashrc(Bash 쉘)나 .cshrc(C 쉘)등 자신이 사용하는 쉘의 환경 파일에 등록을 해두면 로그인과 동시에 설정이 되어 편리하게 이용할 수 있다.

2. 파일의 내용보기 및 편집

앞에서 파일 목록을 보고, 파일과 디렉토리를 통째로 복사하거나 이리저리 옮기고 삭제하는 방법을 설명했다. 이번에는 파일 자체의 내용을 보고, 편집하는 방법을 배워보자.

2-1. 파일의 내용 보기와 리다이렉트(redirect) cat, >,<,>>

cat은 파일 혹은 다른 출력 내용을 열람할 때 사용하는 명령어이다. 보잘것없어 보이는 프로그램이지만 >,< 등의 리다이렉터(redirector)와 함께 사용하면 거의 모든 작업에 사용될 수 있을 만큼 강력한 명령어가 된다. cat의 사용 방법은 다음과 같다.

```
cat [옵션] [파일]
```

```
[namul@dcclab namul]$ cat
It's the first line.
It's the first line.
And it's the second line.
And it's the second line.
Last it's the third line.
Last it's the third line.
^d
[namul@dcclab namul]$ _
```

^d는 ctrl과 d를 함께 누르라는 뜻이다. 유닉스에서 ^d를 입력하면 EOF(End Of File:

파일의 끝)라는 특별한 문자가 입력되는 경우가 많다. 즉 ^d를 눌러 EOF를 입력하면 그 줄에서 입력이 끝나는 것이다. cat를 입력한 후 그 다음 줄부터 입력하는 내용은 그대로 화면에 다시 한번씩 나타날 것이다. cat이라는 명령만을 내리면 표준 입력으로 입력받은 내용을 표준 출력으로 내보내게 된다. 보통 키보드를 통해 입력하는 것이 표준 입력이 되고, 화면으로 출력하는 것이 표준 출력이 된다. 따라서 입력한 내용이 그대로 다시 한번씩 나타나게 되는 것이다.

```
[namul@dcclab namul]$ cat > test
It is the first line.
And it's the second line.
Last it's the third line.
^d
[namul@dcclab namul]$ ls
test
[namul@dcclab namul]$ _
```

앞의 예에서는 >를 사용하여 출력 내용을 표준 출력으로 보내는 대신 test라는 파일로 리다이렉트(redirect)시킨 것이다. 즉 입력한 내용이 test라는 파일에 저장된다(ls로 확인할 수 있다). 이번에는 생성된 파일 내용을 확인해보자.

```
[namul@dcclab namul]$ cat < test
It is the first line.
And it's the second line.
Last it's the third line.
[namul@dcclab namul]$ _
```

물론 cat < test 대신에 cat test라고 입력해도 된다. 이번에는 <를 이용해 표준 입력 대신 test라는 파일을 사용하도록 입력 방향을 리다이렉트시킨 것이다. 즉 출력을 리다이렉트할 때에는 >, 입력을 리다이렉트 할 때에는 <를 사용한다. 다음과 같이 입력해보자.

```
[namul@dcclab namul]$ cat < test > test2
[namul@dcclab namul]$ ls
test  test2
[namul@dcclab namul]$ cat test2
It is the first line.
And it's the second line.
Last it's the third line.
[namul@dcclab namul]$ _
```

cat < test > test2는 test라는 파일을 입력으로 받아 test2라는 파일로 보내라는 뜻이다. 물론 cat test >test2 또는 cat > test2 < test로 써도 된다.

출력을 리다이렉트하는 리다이렉터로 > 외에 >>도 자주 사용된다. >는 기존 파일을 지우고 새 파일을 만드는 데 반해 >>는 기존 파일의 뒤에 내용을 덧붙인다.

리다이렉터는 cat 외에도 거의 모든 Linux 명령어와 함께 사용될 수 있다.

```
[namul@dcclab namul]$ ls -l  > list
[namul@dcclab namul]$ cat list
```

ls -l > list 는 ls라는 명령의 결과를 표준 출력으로 보내는 대신 list라는 파일에 저장한 것이다.

2-2. 큰 파일의 내용보기 more, less, head, tail, 파이프(|)

앞의 예에서와 같이 작은 파일이라면 cat만으로도 불편함이 없지만 파일 내용이 한 화면이 넘을 때에는 cat으로는 불편하다. 이럴 경우 사용할 수 있는 명령으로 more와 less가 있다. more는 전통적인 유닉스의 명령어로, 다음과 같이 출력 내용을 한 화면씩 보여준다. 각각의 사용법은 다음과 같다.

```
more [옵션] [파일]
less  [옵션] [파일]
```

다음과 같이 /etc/services라는 파일의 내용을 확인해보자.

```
[namul@dcclab namul]$ more /etc/services
# /etc/services:
# $Id: services,v 1.31 2002/04/03 16:53:20 notting Exp $
#
# Network services, Internet style
#
# Note that it is presently the policy of IANA to assign a single well-known
# port number for both TCP and UDP; hence, most entries here have two entries
# even if the protocol doesn't support UDP operations.
# Updated from RFC 1700, ``Assigned Numbers'' (October 1994).  Not all ports
# are included, only the more common ones.
#
# The latest IANA port assignments can be gotten from
#       http://www.iana.org/assignments/port-numbers
# The Well Known Ports are those from 0 through 1023.
# The Registered Ports are those from 1024 through 49151
# The Dynamic and/or Private Ports are those from 49152 through 65535
#
# Each line describes one service, and is of the form:
#
# service-name  port/protocol  [aliases ...]   [# comment]

tcpmux          1/tcp                           # TCP port service multiplexer
tcpmux          1/udp                           # TCP port service multiplexer
rje             5/tcp                          # Remote Job Entry
rje             5/udp                           # Remote Job Entry
echo            7/tcp
echo            7/udp
discard         9/tcp           sink null
discard         9/udp           sink null
systat          11/tcp          users
systat          11/udp          users
daytime         13/tcp
daytime         13/udp
qotd            17/tcp          quote
--More--(5%)
```

space bar를 누르면 다음 화면으로 넘어간다. 끝까지 보지 않고 중간에 끝내려면 q를 누르면 된다. 전통적인 유닉스에는 없지만 Linux에는 more보다 더 편리한 less라는 명령어가 있다. more는 앞 방향으로만 진행할 수 있지만 less에는 뒤로 돌아가는 기능이 있다.

less에서 f 또는 page down 키를 누르면 다음 화면으로 넘어갈 수 있고, b 또는 page up 키로 다시 이전 화면으로 돌아갈 수 있다. 뿐만 아니라 위아래 화살표나 j, k를 이용해 한 줄씩 스크롤 하는 것도 가능하다. less를 끝낼 때에는 q를 누르면 된다. less는 기본적으로 한글을 제대로 출력하지 못한다. 한글이 포함되어 있는 내용을 less로 보려면 -r 옵션을 사용해야 한다. 물론 less를 자주 사용할 경우 alias를 지정해두면 편리하다.

파일 전체를 한 페이지씩 보여주는 more와 less 외에 파일의 시작 부분만을 보여주는 head와 파일의 끝부분만을 보여주는 tail이라는 명령어가 있다. 사용법은 다음과 같다.

```
head [옵션] [파일]
tail  [옵션] [파일]
```

```
[namul@dcclab namul]$ head /etc/services
# /etc/services:
# $Id: services,v 1.31 2002/04/03 16:53:20 notting Exp $
#
# Network services, Internet style
#
# Note that it is presently the policy of IANA to assign a single well-known
# port number for both TCP and UDP; hence, most entries here have two entries
# even if the protocol doesn't support UDP operations.
# Updated from RFC 1700, ``Assigned Numbers'' (October 1994).  Not all ports
# are included, only the more common ones.
[namul@dcclab namul]$
```

head와 tail은 -n 옵션을 사용하여 보여줄 줄(line) 수를 지정할 수 있다.

```
[namul@dcclab namul]$ head -n 5 /etc/services
[namul@dcclab namul]$ head -n 5 /etc/services
# /etc/services:
# $Id: services,v 1.31 2002/04/03 16:53:20 notting Exp $
#
# Network services, Internet style
#
[namul@dcclab namul]$
```

more와 less는 단독으로 사용되기도 하지만 |와 함께 사용되기도 한다. |는 파이프라고 읽는데, 한 명령의 결과를 다른 명령의 입력으로 보내는 역할을 한다. 예를 들어 디렉토리 안에 파일이 너무 많아 ls -l 명령의 결과가 한 화면을 넘어간다면 다음과 같이 파이프와 more를 이용하여 한 화면씩 나오게 할 수 있다.

```
[namul@dcclab namul]$ ls -l /usr/bin | more
total 213408
-rwxr-xr-x    1 root     root         7835 Jun 24  2002 411toppm
-rwxr-xr-x    1 root     root           82 Jun 23  2002 4odb
-rwxr-xr-x    1 root     root           87 Jun 23  2002 4rdf
-rwxr-xr-x    1 root     root           81 Jun 23  2002 4xslt
-rwxr-xr-x    1 root     root           99 Jun 23  2002 4xupdate
-rwxr-xr-x    1 root     root         1853 Aug 11  2002 AbiWord
-rwxr-xr-x    1 root     root        84800 Aug 28  2002 CDDBSlave2
-rwxr-xr-x    1 root     root       252375 Aug 27  2002 DataManager
-r-xr-xr-x    1 root     root        14361 Aug  7  2002 GET
-r-xr-xr-x    1 root     root        14361 Aug  7  2002 HEAD
-rwxr-xr-x    1 root     root         1406 Mar 15  2003 Magick++-config
-rwxr-xr-x    1 root     root         1265 Mar 15  2003 Magick-config
lrwxrwxrwx    1 root     root           14 Apr 11  2003 Mail -> ../../bin/mail
-rwxr-xr-x    1 root     root          437 Aug 29  2002 MakeTeXPK
-rwxr-xr-x    1 root     root         7614 Aug 27  2002 ODBCConfig
--More--
```

파이프 |는 앞에서 설명한 리다이렉터 >,<과 비슷한 것 같지만 둘은 분명히 다르다. 리다이렉터가 입력이나 출력을 표준 입력과 출력을 파일로 보내는 데 반해 파이프는 한 명령의 출력을 다른 명령의 입력으로 보낸다. 리다이렉터와 마찬가지로 파이프 역시 head와 tail을 비롯해 거의 모든 Linux 명령과 함께 사용될 수 있다. 유닉스나 Linux를 오랫동안 사용한 사람들 중에는 다음과 같이 5~6개의 명령어를 파이프와 리다이렉터로 연결하여 사용하는 사람이 많다.

```
[namul@dcclab namul]$ ifconfig lo | head -n 2 | tail -n 1| cut -d: -f2 > ipoflo
[namul@dcclab namul]$ cat ipoflo
127.0.0.1          Mask
[namul@dcclab namul]$ _
```

위의 명령에 대해서는 man 페이지를 참고한다.

2-3. 텍스트 파일 편집하기 vi, awk, sed

2-3-1. vi

이제 파일을 편집해볼 차례이다. Linux에서 파일 편집의 기본은 vi이다. vi의 사용법만으로도 간단한 별책 부록 한 권은 너끈히 채울 수 있는 분량이 되기 때문에 여기서는 기본적인 사용법에 대해서만 알아보겠다. 좀더 자세한 사용 방법은 '부록 II의 vi 편집기 명령어 요약'을 참고하기 바란다. 우선 vi를 실행하는 방법은 다음과 같다.

```
vi [옵션] [파일]
```

[파일]을 생략하면 그냥 비어 있는 상태로 vi가 실행된다. 이전에 만들었던 test 파일을 vi로 열어보자.

```
[namul@dcclab namul]$ vi  test
```

다음과 같은 화면을 볼 수 있다.

```
It's the first line
It's the second line
Last, it's the third line
~
~
~
"test" 3L, 67C
```

:q 라고 입력한 후 Enter를 눌러보자.

```
It's the first line
It's the second line
Last, it's the third line
~
~
:q
[namul@dcclab namul]$ _
```

그렇다. :q는 vi를 빠져나가는 명령어이다. 이제 다시 vi를 실행해보자. 이번에는 뒤에 편집할 파일을 적지 말고 vi만 입력한 후 Enter를 눌러보자.

vi의 가장 큰 특징은 입력모드와 명령모드가 구분되어 있다는 점이다. vi를 실행하면 보통 기본은 명령모드이다. 이 상태에서 키보드로 입력하는 글자 하나하나는 화면에 입력되는 것이 아니라 하나의 명령어가 된다. 파일에 어떤 내용을 입력하기 위해서는 먼저 명령모드에서 입력모드로 들어가는 명령을 입력해 입력모드로 전환한 후 내용을 입력해야 한다. 그러한 명령은 여러 가지가 있는데 그 중에서 가장 자주 사용되며, 기본이 되는 것은 a와 i이다. a를 누른 후 내용을 입력하면 커서가 위치하고 있던 자리의 바로 다음 자리부터 내용이 입력되고, i를 누른 후 내용을 입력하게 되면 커서가 위치하고 있던 자리에서부터 내용이 입력된다. vi에서 다음과 같이 입력한다.

```
It is the second line,<엔터>It's the last line.<엔터>And it it the second line.
```

화면에는 다음과 같은 내용이 입력된다. 밑줄은 커서를 나타낸다.

```
It's the second line.
It's the last line.
And it is the second line.
~
~
~
```

그렇다. 맨 앞의 a는 명령모드에서 입력모드로 전환하는데 사용되었기 때문에 입력되지 않았다. 입력을 마쳤으면 Esc 키를 눌러 다시 명령모드로 전환한다. Esc를 누르면 어떤 모드에서건 항상 명령모드로 전환된다. 이제 입력한 내용을 수정해보자. 요즘 사용되는 대

부분의 vi에서는 화살표 키도 사용 가능하지만 원래 vi는 커서 이동에 h(←),j(↓),k(↑),l(→)을 이용한다. 바로 오른쪽 손가락들이 놓이는 위치이기 때문에 익숙해지면 화살표 키를 누르는 것보다 훨씬 편리하다. 자꾸 눌러봐서 익숙해지는 것이 좋다.

이제 첫 번째 줄 second의 첫 글자 s에 커서를 위치시킨 후 x를 몇 번 눌러본다. 다음과 같이 글자가 하나씩 지워진다.

```
It's the _line.
It's the last line.
And it is the second line.
~
~
~
```

i를 눌러 입력모드로 전환한 후 first를 입력하고 다시 Esc를 눌러 명령모드로 전환한다.

```
ifirst[Esc]
```

```
It's the first_line.
It's the last line.
And it is the second line.
~
~
~
```

이번에는 두 번째 줄을 통째로 지워보자. 커서를 두 번째 줄에 위치시킨 후 dd를 입력하면 다음과 같이 두 번째 줄이 통째로 지워진다.

```
It's the first line.
And it is the second line.
~
~
~
```

지워진 내용은 버퍼라는 기억 장소에 임시 저장된다. p를 누르면 다음과 같이 된다.

```
It's the first line.
And it is the second line.
It's the last line.
~
~
~
```

지운 내용을 커서의 앞쪽에 끼워 넣는 명령은 P, 뒤쪽에 끼워 넣는 명령은 p이다. vi는 대문자와 소문자 명령이 서로 다르다. 대문자 명령은 shift 키와 함께 누르라는 뜻이다. 이번에는 커서를 첫 번째 줄로 이동시킨 후 J를 눌러보자. J는 두 줄을 하나로 합치는 명령어이다.

```
It's the first line._And it is the second line.
It's the last line.
~
~
~
```

이제 앞에서 배운 :q를 이용해 vi를 빠져나가자.

```
It's the first line._And it is the second line.
It's the last line.
~
~
:q
```

어, 그런데 vi가 다음과 같은 메시지를 보낸다.

```
It's the first line._And it is the second line.
It's the last line.
~
~
E37: No write since last change (use ! to override)
```

저장을 하지 않았으니 그냥은 빠져나갈 수 없다는 메시지이다. 편집한 파일을 저장하는 방법을 모르면 아무 소용이 없지 않은가! 편집한 파일을 저장하는 명령은 :w이다. vi가 반항을 해도 그냥 저장하지 않고 빠져나가려면 !를 붙여서 :q!라고 입력하면 된다. !는 그냥 밀어붙이라는 뜻이다. 다음과 같이 입력해보자.

```
It's the first line._And it is the second line.
It's the last line.
~
~
:w test
```

이번에는 다음과 같은 메시지가 나타날 것이다.

```
~
~
E13: File exists (use ! to override)
```

test라는 파일이 이미 존재하기 때문에 같은 이름으로는 저장할 수 없다는 뜻이다. 파일을 저장할 때 역시 이미 존재하는 파일을 덮어쓰도록 밀어붙이려면 !를 붙여서 :w!를 입력하면 된다.

vi에서 :q나 :w처럼 명령모드에서 :로 시작되는 명령어가 많이 있다. 명령모드에서 :를 입력하면 화면 맨 아래에 :가 나타나면서 내용이 입력되는데, 이런 상태를 따로 끝줄모드라고 구분하여 부르기도 한다. 문서를 저장한 후 끝마치는 작업을 한 번에 하려면 :wq를 입력하면 된다. 물론 이때에도 반항하면 :wq!로 밀어붙일 수 있다.

이 정도면 간단한 설정 파일 편집 정도는 할 수 있을 것이다. 지금까지 설명한 내용은 vi의 '기초 중의 기초'이다. vi는 웬만한 다른 편집기에 있는 기능은 다 가지고 있다. 보다 자세한 설명은 부록을 참고한다.

2-3-2. awk

텍스트 파일을 생성하고 수정하는 사람에게 awk와 sed는 편집에 유용한 툴이다. 대화식 문서 편집기로 할 수 있는 일이라 하더라고 sed와 awk를 이용하면 반복적인 것을 더 짧은 시간에 처리할 수 있다. 이 두 프로그램은 또한 편집 스크립트를 제공하며 필요에 맞게 스크립트를 작성하여 여러 가지 일을 할 수 있다.

awk는 일종의 프로그래밍 언어지만 일반적인 언어라기보다는 주로 패턴 검색과 조작을 주목적으로 만들어진 것이다. 파일의 각 줄에서 필드(field)를 인식할 수 있는 패턴 매칭 기능을 가지고 있어서 이들 필드를 자유자재로 조작 가능한 유틸리티를 작성하고자 만들어졌다. awk의 가장 일반적인 사용법은 다음과 같다.

```
% awk ' 패턴 { 액션 }
        패턴 { 액션 }
        …
            ' [파일]
% awk -f 스크립트 파일 [파일]
```

명령행에 곧바로 스크립트를 지정할 수 있고 -f 옵션을 사용하여 스크립트 파일에 스크립트를 저장할 수도 있다.

명령행에서 awk를 사용하는 예를 보자. "coins.txt"라는 파일은 각 라인은 다음의 항목들로 구성된다.

```
metal   weight in ounces   date minted   country of origin   description
```

파일은 다음과 같다.

```
gold      1     1986   USA                American Eagle
gold      1     1908   Austria-Hungary    Franz Josef 100 Korona
silver   10     1981   USA                ingot
gold      1     1984   Switzerland        ingot
gold      1     1979   RSA                Krugerrand
gold      0.5   1981   RSA                Krugerrand
gold      0.1   1986   PRC                Panda
silver    1     1986   USA                Liberty dollar
gold      0.25  1986   USA                Liberty 5-dollar piece
silver    0.5   1986   USA                Liberty 50-cent piece
silver    1     1987   USA                Constitution dollar
gold      0.25  1987   USA                Constitution 5-dollar piece
gold      1     1988   Canada             Maple Leaf
```

다음과 같이 하여 이 파일에서 금화만을 뽑아낼 수 있다.

```
% awk '/gold/' coins.txt
```

awk는 파일에서 gold가 있는 줄을 찾아서 출력하게 된다.

```
gold    1     1986  USA                American Eagle
gold    1     1908  Austria-Hungary    Franz Josef 100 Korona
gold    1     1984  Switzerland        ingot
gold    1     1979  RSA                Krugerrand
gold    0.5   1981  RSA                Krugerrand
gold    0.1   1986  PRC                Panda
gold    0.25  1986  USA                Liberty 5-dollar piece
gold    0.25  1987  USA                Constitution 5-dollar piece
gold    1     1988  Canada             Maple Leaf
```

awk은 grep이나 find 같은 기능을 가지고 있다.

```
% awk '/gold/ {print $5, $6, $7, $8}' coins.txt
```

.위의 명령에서 {print $5,$6,$7,$8}은 5,6,7,8번째의 필드나, 필드 변수를 출력하라는 것이다. 결과는 다음과 같다

```
American Eagle
Franz Josef 100 Korona
ingot
Krugerrand
Krugerrand
Panda
Liberty 5-dollar piece
Constitution 5-dollar piece
Maple Leaf
```

coins.txt 파일에서 각 필드 변수는 파일의 각 라인의 다음과 같이 매치된다.

```
metal:        $1
weight:       $2
date:         $3
country:      $4
description:  $5 through $8
```

그러면 이제, coins.txt파일에서 1980년 이전에 주조된 동전만 골라내보자.

```
% awk '{if ($3 < 1980) print $3, "    ", $5,$6,$7,$8}' coins.txt
```

위 명령은 세 번째 필드, 즉 date가 1980보다 작은 라인을 출력하는데, 이때 3번째 필드와, " "의 공백, 그리고 5~8번째 필드를 출력하라는 것이다. 아래 결과를 보인다.

```
1908    Franz Josef 100 Korona
1979    Krugerrand
```

awk는 BEGIN과 END를 사용하는 확장된 형태를 제공하고 있다.

```
% awk 'BEGIN { 초기화 }
        검색패턴 { 액션 }
        검색패턴 { 액션 }
        ...
       END' { 종료 액션 }' [파일]
```

BEGIN은 파일을 읽어 들이기 전에 요구되는 초기화를 수행하는 구문이며, END는 awk가 모든 입력을 처리한 후, 수행할 구문을 명시한다. 다음의 예를 보자.

```
% awk '/gold/ {ounces += $2} END {print "value = $" 425*ounces}' coins.txt
```

이 예제에서 "ounces"는 사용자가 정의한 변수이다. conis.txt 파일에서 gold가 포함된 라인의 무게를 모두 더해서 ounces 변수에 넣고, 425*ounces의 값을 출력한다. 다음의 결과가 출력된다.

```
value = $2592.5
```

2-3-3. sed(Stream Editor)

sed는 스트림 편집기(Stream Editor)의 약자로서 파일의 수정을 주목적으로 한다. 파일을 순방향으로 읽는 동안 연산을 수행하며 텍스트 파일에서의 반복 수정에 용이하다.

sed 명령어는 1개 라인씩 입력 라인을 읽어 들여 표준 출력에 출력한다. 일치하는 문자열이 있으면 그 문자열들을 대치한 후 출력하고 일치하는 문자열이 없으면 그 라인은 수정되지 않고 그대로 출력된다. 가장 기본적인 사용법은 다음과 같다.

```
% sed 's/찾는문자열/바꿀문자열' 입력파일
```

입력파일에서 문자열을 찾아 바꿀 문자열로 치환해주며, 결과는 표준출력으로 보내진다.

```
% sed 's/old/new' myfile
```

위의 예는 myfile에서 "old" 문자열을 찾아서 "new" 문자열로 바꾼다. 파일명이 주어지면 sed는 표준 입력 대신에 파일로부터 읽는다. 만약 하나 이상의 편집 명령어를 실행하고자 한다면 -e 옵션을 사용한다.

```
% sed -e 's/old/new' -e '/bad/d' myfile
```

myfile의 처음부터 끝까지 "old"를 찾아서 "new"로 바꾸고, "bad"를 찾아서 모두 삭제한다.

```
% sed -f scriptfile myfile
```

명령어들을 파일에 넣고 -f 옵션을 주어 sed가 그 파일을 읽도록 할 수도 있다.

sed 명령어들은 기본적으로 모든 라인에 적용이 된다. sed는 두 개까지 라인 지정을 할 수 있으며 라인 번호, 라인 지정 심볼, 정규표현 식을 이용하여 라인을 지정한다. 라인 지정이 없으면 명령어는 각각의 라인에만 적용된다. 라인 지정이 하나 있다면 명령어는 지정된 라인에만 적용되고, !가 따라오면 명령어는 지정된 라인에 매치되지 않는 모든 라인에

적용된다.

```
% sed 'd' myfile          // 모든 라인이 지워진다.
% sed '3d' myfile         // 세 번째 라인이 지워진다.
% sed '$d' myfile         // 마지막 라인이 지워진다.
```

정규표현 식은 슬래시(/)로 감싸져야 한다. 다음은 삭제 명령이다.

```
% sed '/^$/d' myfile
```

위의 경우는 모든 공백 라인을 지우며, 다른 라인은 건드리지 않는다.
일상적인 sed 명령으로 다음과 같은 것이 있다.

```
% sed 's/xx/yy/g' myfile      // 모든 라인을 치환한다.
% sed '/BSD/d' myfile                  // BSD를 포함하는 모든 라인을 삭제한다.
% sed '1,10d' myfile                   // 1라인부터 10라인까지 지운다.
% sed '/^BEGIN/,/^END/p' myfile  // BEGIN과 END 사이의 모든 라인을 출력한다.
% sed '/SAVE/!d' myfile   // SAVE를 포함하고 있지 않은 모든 라인을 삭제한다.
```

2-4. 파일 찾기 및 내용 찾기 find, grep

저장되어 있는 파일이 몇 개 되지 않는다면 문제가 없겠지만 만약 어떤 파일이 어떤 디렉토리에 들어 있는지 알 수 없다면 find 명령어를 이용하여 파일이 어디에 있는지 찾을 수 있다. 파일의 크기나 소유자 등 여러 가지 방법으로 찾을 수 있지만 파일의 이름으로 찾는 방법만큼 많이 사용되지는 않는다. find를 사용해 특정 이름의 파일을 찾을 때의 방법은 다음과 같다.

```
find [디렉토리] -name [파일명]
```

앞의 명령과 같이 입력하면 [디렉토리]에 적어준 디렉토리에서부터 그 하위 디렉토리에서 [파일명]이라는 이름의 파일이 있는지 모두 검색한다.

만약 시스템 전체에서 찾고 싶다면 [디렉토리]에 루트 디렉토리, 즉 /를 적어주면 된다. [디렉토리]를 생략하면 현재 작업 중인 디렉토리와 그 하위 디렉토리를 검색한다. 다음과 같이 현재 디렉토리에서 services라는 파일을 검색해보자.

```
[namul@dcclab namul]$ find -name services
./services
```

이번에는 특정 파일 내에서 어떤 내용이 있는지를 검색하는 방법에 대해서 알아보자. 이때 사용하는 명령어는 grep이다.

```
grep [옵션] [패턴] [파일]
```

services라는 파일에 http라는 단어가 있는지 검색해보자.

```
[namul@dcclab namul]$ grep http services
www             80/tcp          http            # WorldWideWeb HTTP
https           443/tcp                          # MCom
https           443/udp                          # MCom
[namul@dcclab namul]$ _
```

앞에서와 같이 grep 명령을 사용하면 [패턴]을 포함하고 있는 줄을 표시해준다. -v 옵션을 주는 경우 [패턴]을 포함하고 있는 줄 대신 포함하고 있지 않은 줄을 보여준다.

3. 파일과 디렉토리의 압축과 해제

윈도우에서 파일을 압축하고 해제할 때에는 WinZIP이나 WinRAR, 혹은 알집 같은 프로그램을 사용한다. Linux에서 파일과 디렉토리를 압축할 때에는 tar와 gzip이라는 프로그램을 자주 사용한다. 한 가지 특이한 점은 Linux에서는 여러 개의 파일을 한데 묶는 프로그램과 압축하는 프로그램이 따로 있다는 점이다.

3-1. 여러 개의 파일을 하나로 묶기 tar

Linux에서 여러 개의 파일을 하나로 묶을 때에는 tar를 이용한다.

```
tar [옵션] [묶인 파일] [묶을 파일]
```

[묶인 파일]이란 여러 개의 파일을 한데 묶어 새로 만들어지는 파일을 의미하며, 보통 확장자로 .tar를 사용한다. [묶을 파일]은 한데 합친 파일로, 여러 개의 파일을 한꺼번에 나열할 수 있으며, 디렉토리를 적어줄 수도 있다. 만약 디렉토리를 적어줄 경우 그 디렉토리 안에 있는 파일까지 모두 한꺼번에 묶인다.

tar에서 자주 사용하는 옵션은 다음과 같다.

-c	[묶인 파일]을 새로 만들어 [묶을 파일]을 묶는다.
-r	[묶인 파일]에 [묶을 파일]을 덧붙여서 묶는다.
-x	[묶인 파일]을 푼다
-v	파일을 묶거나 풀면서 파일 하나하나를 화면으로 보여준다.
-f [HOSTNAME:]F	아카이브 파일 또는 디바이스 F를 사용한다.

앞에 나열한 것은 가장 자주 사용되는 옵션이다. 그러나 이 옵션은 한꺼번에 사용되는 경우가 많으므로 묶을 때는 -cvf, 풀 때는 -xvf 이런 식으로 한꺼번에 기억해두는 것이 편리하다. 이제 직접 파일을 묶고 풀어보자.

```
[namul@dcclab namul]$ ls
test   test2
[namul@dcclab namul]$ tar -cvf test.tar *
test
test2
[namul@dcclab namul]$ ls
test    test.tar    test2
[namul@dcclab namul]$ _
```

현재 디렉토리의 모든 파일을 test.tar라는 파일로 묶은 것이다. 이번에는 원래 있던 파일을 삭제한 후 test.tar라는 파일을 다시 풀어보자.

```
[namul@dcclab namul]$ ls
test    test.tar    test2
[namul@dcclab namul]$ rm test test2
[namul@dcclab namul]$ ls
test.tar
[namul@dcclab namul]$ tar -xvf test.tar
test
test2
[namul@dcclab namul]$ ls
test    test.tar    test2
[namul@dcclab namul]$ _
```

이제 어느 정도 tar가 무슨 일을 하는지 감이 올 것이다.

3-2. 파일의 압축과 해제 gzip, gunzip

앞에서 묶는 파일을 압축하는 방법을 알아보자. Linux에서 압축의 표준은 gzip이라고 해도 과언이 아닐 만큼 gzip은 많이 사용된다. 파일을 압축할 때에는 gzip, 반대로 압축을 풀 때에는 gunzip 명령어를 사용한다. 사용 방법은 다음과 같다.

```
gzip [옵션] [파일]
gunzip [옵션] [파일]
```

tar의 경우 옵션을 생략하면 안 되지만 gzip이나 gunzip의 경우 보통 [옵션]은 생략하고 사용하는 경우가 많다. 다음 화면을 보자.

```
[namul@dcclab namul]$ ls -l
total 20
-rw-rw-r--    1 namul    namul          67 Feb  9 14:26 test
-rw-rw-r--    1 namul    namul       10240 Feb  9 14:26 test.tar
-rw-rw-r--    1 namul    namul          67 Feb  9 14:26 test2
[namul@dcclab namul]$ gzip test.tar
[namul@dcclab namul]$ ls -l
total 12
-rw-rw-r--    1 namul    namul          67 Feb  9 14:26 test
-rw-rw-r--    1 namul    namul         196 Feb  9 14:26 test.tar.gz
-rw-rw-r--    1 namul    namul          67 Feb  9 14:26 test2
[namul@dcclab namul]$ _
```

gzip을 사용해 test.tar라는 파일을 압축하는 화면이다. 앞의 화면을 보면, 10240바이트 파일이 196바이트로 크기가 줄어든 것을 확인할 수 있다. gzip으로 압축된 파일의 확장자는 보통 .gz이다. tar로 묶은 파일의 확장자는 .tar이므로 tar로 묶은 후 gzip으로 압축한 파일은 확장자가 .tar.gz이 된다. 이를 짧게 줄여서 .tgz라는 확장자를 사용하기도 한다.

gzip을 사용할 때 한 가지 알아두어야 할 점은 gzip을 사용하여 파일을 압축할 경우 원본 파일이 자동으로 지워진다는 점이다. 앞의 화면을 보면 test.tar라는 원래의 파일이 없어진 것을 알 수 있다. gzip으로 압축된 파일을 다시 풀 때에는 gunzip을 이용한다.

```
[namul@dcclab namul]$ ls
test    test.tar    test2
[namul@dcclab namul]$ gunzip test.tar.gz
[namul@dcclab namul]$ ls
test.tar
[namul@dcclab namul]$ tar -xvf test.tar
test    test.tar    test2
[namul@dcclab namul]$ _
```

gzip에서도 다른 명령어처럼 와일드카드를 사용할 수 있는데, 와일드카드를 사용할 경우에는 여러 개의 파일이 한꺼번에 압축되는 것이 아니라 다음과 같이 각각의 파일이 따로따로 압축된다.

```
[namul@dcclab namul]$ ls
test    test.tar    test2
[namul@dcclab namul]$ gzip *
[namul@dcclab namul]$ ls
test.gz    test.tar.gz    test2.gz
[namul@dcclab namul]$ gunzip *
[namul@dcclab namul]$ ls
test    test.tar    test2
[namul@dcclab namul]$ _
```

gunzip을 이용해 압축을 풀 때 역시 마찬가지이다. 그런데 윈도우에서 사용하는 WinZIP이나 알집 같은 프로그램에 익숙한 사용자는 이렇게 파일을 묶은 후 다시 압축하는 것은 좀 번거로워 보일 것이다. 파일을 묶고 압축하는 작업을 한꺼번에 하려면 tar에 단순히 -z 옵션을 붙여주면 된다.

```
[namul@dcclab namul]$ tar -cvfz test.tgz *
test
test2
[namul@dcclab namul]$ ls
test    test.tgz    test2
[namul@dcclab namul]$ _
```

마찬가지로 압축된 파일의 압축을 해제하여 묶음을 풀 때에는 다음과 같이 -xvfz 옵션을 사용하면 된다.

```
[namul@dcclab namul]$ rm test test2
[namul@dcclab namul]$ ls
test.tgz
[namul@dcclab namul]$ tar -xvfz test.tgz
test
test2
[namul@dcclab namul]$ ls
test    test.tgz    test2
[namul@dcclab namul]$ _
```

gzip보다 훨씬 최근에 나온 압축 프로그램으로는 bzip2가 있는데 gzip에 비해 훨씬 뛰어

난 압축률을 자랑한다. tar는 gzip으로 압축하는 옵션 외에 bzip2로 압축하는 옵션도 제공하는데 이 옵션을 사용하기 위해서는 물론 bzip2가 설치되어 있어야 한다.

파일을 묶은 후 bzip2로 압축할 때에는 단순히 -z 대신에 -j 옵션을 사용하면 된다. bzip2의 사용법은 gzip과 거의 유사하기 때문에 따로 설명하지는 않겠다. 파일을 압축할 때에는 bzip2, 압축을 해제할 때에는 bunzip2 명령어를 사용하며, 압축된 파일의 확장자는 보통 .bz2이다.

3-3. 압축된 문서를 편리하게 보기 zcat

Linux 매뉴얼을 비롯한 많은 문서는 gzip으로 압축되어 있다. 이 문서를 일일이 gunzip으로 압축을 풀어서 본 후 다시 gzip으로 압축해놓아야 한다면 상당히 불편할 것이다. 이런 불편을 없애기 위해 압축된 문서를 자동으로 풀어서 보여주는 몇 가지 툴이 있다. 앞에서 파일 내용을 열람하는데 사용했던 cat을 기억하고 있을 것이다. 만약 압축된 파일 내용을 보고 싶다면 이 cat 대신 zcat을 이용하면 된다.

```
[namul@dcclab namul]$ zcat test.gz
It is the first line.
And it is the second line.
It's the last line.
[namul@dcclab namul]$ _
```

4. 멀티태스킹

멀티태스킹이란 쉽게 말해 한 번에 두 가지 이상의 작업을 하는 것을 말한다. 윈도우도 멀티태스킹을 지원하기는 하지만 상당히 불안정하다. 윈도우2000에서는 좀 나아졌다고는 하지만 윈도우 사용자 치고 한 프로그램이 다운되면서 시스템 전체가 다운되는 경험을 해보지 않은 사람은 없을 것이다. 리눅스에서는 대부분 한 프로그램이 잘못되더라도 그 프로그램에 해당되는 프로세스를 찾아내어 죽이면(kill) 해결된다.

4-1. 백그라운드로 작업하기 jobs, fg, bg, kill

이제 두 가지 이상의 작업을 동시에 하는 방법을 배워보자. 먼저 vi를 실행하면 앞에서와 같은 vi의 초기 화면이 나온다. 그 상태에서 ctrl +z를 누른다.

```
[1]+ Stopped                 vi
[namul@dcclab namul]$ _
```

앞에서와 같이 다시 쉘 프롬프트가 나타난다. 프롬프트 위의 [1]+ Stopped vi는 앞에서 실행했던 vi라는 작업이 중지되었다는 뜻이다. 이번에는 more로 /etc/services 파일을 열람한 후 ctrl+z를 누른다.

```
[2]+ Stopped                 more /etc/services
[namul@dcclab namul]$ _
```

지금까지 두 개의 작업을 중지시켰다. 한 가지 더 해보자. 이번에는 find / -name a* > filelist를 입력한 후 곧바로 ctrl+z를 입력한다. a로 시작하는 모든 파일을 찾아 결과를 filelist라는 파일에 저장하라는 명령이다.

```
[3]+ Stopped                 find / -name a* > filelist
[namul@dcclab namul]$ _
```

중지되어 있는 작업을 확인해보는 명령어는 jobs이다. 다음과 같이 입력한다.

```
[namul@dcclab namul]$ jobs
[1]+ Stopped                 vi
[2]- Stopped                 more /etc/services
[3]+ Stopped                 find / -name a* > filelist
[namul@dcclab namul]$ _
```

이 상태에서 다시 중지된 작업으로 돌아갈 때에는 fg를 사용한다. fg는 Front Ground의 약자이다 fg의 사용법은 다음과 같다.

```
fg [%작업번호]
```

[%작업번호]는 jobs를 했을 때 Stopped의 앞에 나오는 번호이다. 생략할 경우 가장 최근에 중단시킨 작업으로 되돌아간다. 가장 최근에 중단시킨 작업에는 앞의 화면에서처럼 +표시가 있어 쉽게 구분할 수 있다. fg %2를 입력해보자. 다시 more의 화면이 나온다.

```
[namul@dcclab namul]$ fg %2
more /etc/services
--More--(4%)
```

space bar를 몇 번 눌러 more를 끝낸 후 다시 한번 jobs를 실행한다.

```
[namul@dcclab namul]$ jobs
[1]- Stopped                 vi
[3]+ Stopped                 find / -name a* > filelist
[namul@dcclab namul]$ _
```

이번에는 fg 대신에 bg를 사용해보자. bg는 Back Ground의 약자로 사용법은 fg와 동일하다.

```
bg [%작업번호]
```

다음과 같이 입력해보자.

```
[namul@dcclab namul]$ bg %3
[3]+ find / -name a* > filelist &
[namul@dcclab namul]$ jobs
[1]- Stopped                 vi
[3]+ Done                    find / -name a* > filelist
[namul@dcclab namul]$ _
```

bg는 작업을 백그라운드(뒷마당)로 계속하라는 명령어이다. 이 bg 명령을 내리면 화면에는 아무것도 나오지 않지만 하드디스크가 돌아가는 소리가 들릴 것이다. 잠시 후 jobs 명령을 내려보면 앞과 같이 작업이 끝났다는 표시(Done)가 되어 있을 것이다. cat filelist 명령으로 확인해보면 a로 시작하는 파일 목록이 저장되어 있음을 알 수 있다.

작업을 백그라운드로 할 경우 사용자는 또 다른 작업을 할 수 있다. 컴파일처럼 오래 걸리는 작업을 백그라운드로 돌려놓고 그동안 다른 작업을 하면 지루하게 컴파일일 끝날 때까지 아무것도 못하고 기다리지 않아도 된다. 그런데 작업을 백그라운드로 하고 싶을 경우 처음부터 작업을 백그라운드로 돌릴 경우엔 명령의 끝에 &를 붙여주면 된다. 예를 들어 앞 명령의 경우는 find / -name a* > filelist &라고 입력하면 된다.

이번에는 중지시킨 작업을 아예 없애버리는 방법을 알아보자. 이때 사용하는 명령은 kill이다. 이 경우 kill의 사용법 역시 bg나 fg와 비슷하다. bg나 fg와는 달리 작업 번호 외에 프로세스 번호를 이용하는 방법도 있지만 이것은 뒤에서 다시 설명한다. 일단 작업 번호를 사용해서 프로세스를 죽일 경우 그 사용 방법은 다음과 같다.

```
kill [%작업번호]
```

마지막으로 하나 남아 있는 작업인 vi를 끝내보자. 다음과 같이 입력하면 된다.

```
[namul@dcclab namul]$ kill %1
[1]+ Terminated                 vi
[namul@dcclab namul]$ jobs
[namul@dcclab namul]$ _
```

5. 쉘 프로그래밍을 위한 명령어

5-1. 쉘 프로그래밍에 유용한 명령 echo, read, eval, exec, expr

도스를 사용해본 경험이 있는 사람이라면 autoexec.bat라는 파일을 한번쯤은 보았을 것이다. 쉘 프로그래밍(혹 쉘 스크립트)은 이와 마찬가지로 순서대로 실행할 여러 명령어들을 기록하는 것이다. 그리고 단순히 순서대로 실행하는 것이 아닌 프로그래밍이라고 불릴 만한 요소들도 가지고 있다. 이러한 쉘 스크립트를 작성할 때 유용하게 쓰이는 명령어들을 몇 가지 정리해보자.

먼저 echo의 사용법을 보자

```
echo [문자열]
echo $[변수명]
```

echo는 문자열이나 변수의 값을 표준 출력을 통해서 문자열을 출력해준다.

```
echo -n "string to output"
```

위의 명령은 개행 문자를 제거하고 출력한다.

read와 eval은 쉘 내장 명령어이다. 쉘 내장(built-in) 명령어는 따로 실행파일로 있지 않고(쉘 프롬프트 상에서의 명령이 존재하지 않고), 쉘 내부에 존재하는 명령어이다. 대부분 쉘 스크립트 작성 시에 편리함을 위해 존재한다.

read는 사용자에게서 입력을 받아서 변수에 입력하는 명령어이다.

```
read [변수명] = $<
```

위의 예는 표준 입력으로 입력을 받아 변수에 직접 할당을 하는 것이다. 변수 생략 시에 자동으로 'REPLY' 변수가 사용된다. '-p' 옵션으로 prompt도 정할 수 있다. csh에서는 $<

을 사용한다. read 명령은 사용자가 공백 문자에 의해 구분되는 여러 개의 값을 입력하면, read 구문에 명시된 변수들에 차례로 대입한다. 만일 변수의 개수보다 많은 입력이 들어오면, 마지막 변수에 나머지 값을 모두 대입한다. 아래를 보자.

```
[namul@dcclab namul]$ cat > readdata
#!/bin/sh
echo -e "Input two data : "
read first second
echo First : $first
echo Second : $second
^D
[namul@dcclab namul]$ chmod 755 readdata
[namul@dcclab namul]$ readdata
Input two data :
blue dark  <- 사용자 입력 데이타
First : blue
Second : dark
[namul@dcclab namul]$
```

eval 명령은 인수를 평가하게 해준다. 이 명령의 동작은 다음의 짧은 예제를 통해서 가장 분명히 확인할 수 있다.

```
foo=10
x=foo
y='$'$x
echo $y
```

이것은 $foo라는 결과를 출력한다. 그러나 다음은 10을 출력한다.

```
foo=10
x=foo
eval y='$'$x
echo $y
```

그래서 eval은 추가적인 $와 다소 비슷하다. 이것은 변수 값의 값을 준다. eval 명령은 코드를 생성하고 즉시 실행하게 해주므로 매우 유용하다.

exec 명령은 일반적으로 현재 쉘을 다른 프로그램으로 대체하는 데 사용된다.

```
exec wall "Thanks for all the fish"
```

이 실행의 결과는 다음과 같이 쉘에 뿌려진다. wall 명령은 메시지를 브로드 캐스트한다.

```
[namul@dcclab namul]$
Broadcast message from namul (pts/0) (Fri Feb 13 22:23:36 2004):

Thanks for all the fish
```

expr 명령은 수치계산을 수행할 때 사용되는 명령어이다. 연산식을 인수로 받고, 그 결과 값을 반환한다. expr를 사용하기 위해 알아야 할 것은, 인수로 주어지는 연산식을 표현

하는 방법이다. 많은 표현이 쉘이 사용하는 문자와 구분되기 위해서 역슬래쉬 문자(\)와 조합된다. 모든 요소들은 공백 문자로 분리되어야 한다. 그렇지 않으면 모두 하나의 요소로 취급되기 때문에, 원하는 결과를 얻을 수 없다.

```
\* : 곱하기
\/ : 나누기
% : 나머지
+ : 더하기
- : 빼기
=\> \< \<= \>= != : 비교연산자
\& : and 연산자
\| : or 연산자
\( \) : 괄호
length 문자열 : 문자열의 길이를 반환한다.
```

다음의 예를 보자.

```
[namul@dcclab shell_scripts]$ cat > exprex
#!/bin/sh
var=`expr 5 \* 10`
echo $var
echo `expr length Programming!`
^D

[namul@dcclab shell_scripts]$ chmod 755 exprex
[namul@dcclab shell_scripts]$ exprex
50
12
[namul@dcclab shell_scripts]$
```

위에서 `문자는 back quotes(~ 아래의 키)이다. back quotes로 묶여진 문자열은 명령어로 인식되고, 변수에는 명령어의 결과 값이 들어간다.

6. 기타 알아두어야 할 것

6-1. 명령어의 사용법 보기 man, info

지금까지 상당히 많은 명령어를 살펴보았다. 하지만 천재가 아니고서는 각 명령어의 사용법과 옵션을 모두 외우기는 어렵다. 앞에서 가장 먼저 설명한 ls의 옵션이 몇 가지쯤이나 될 거라고 생각하는가? 잘해야 한 20가지쯤 되지 않을까 생각하겠지만 60가지가 넘는다고 한다. 게다가 어떤 옵션에는 --color=yes, --color=no, --color=auto처럼, yes, no, auto 등의 옵션의 옵션까지 있다. 그러니 이 모든 옵션을 외우고 다니는 사람은 없을 것이다. 그래서 사용되는 명령이 바로 man이다. 이것은 어떤 명령어의 맨 페이지(사용법과 옵션이 정리되어 있는 문서)를 보여준다.

```
man [명령어]
```

man man이라고 입력하면 man이라는 명령어로 man 자체의 사용 방법을 볼 수도 있다. 만약 ls의 사용법을 보고 싶다면 다음과 같이 입력하면 된다.

```
[namul@dcclab namul]$ man ls
LS(1)                          User Commands                          LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List  information  about  the FILEs (the current directory by default).
       Sort entries alphabetically if none of -cftuSUX nor --sort.

       Mandatory arguments to long options are  mandatory  for  short  options
       too.

       -a, --all
              do not hide entries starting with .

       -A, --almost-all
              do not list implied . and ..

       --author
              print the author of each file
```

맨 페이지는 기본 페이지를 이용해 보여주도록 되어 있다. less가 설치되어 있다면 less가 기본 페이지가 되고 만약 설치되어 있지 않다면 more가 기본 페이지가 된다. 한글 맨 페이지를 설치하면 몇몇 명령어의 사용법은 한글로 볼 수 있다.

man과 비슷한 명령어로 info가 있다. info는 어떤 프로그램에 대한 정보를 보여준다. 사용법은 man과 비슷하다.

```
info [명령어]
```

info의 키 바인딩은 emacs의 키 바인딩과 비슷하다. info를 빠져나가려면 q를 누르면 된다. 보다 자세한 사용법은 info의 맨 페이지를 통해 알아볼 수 있다.

```
man info
```