유틸리티
Utility

- 압축: tar, gzip
- 텍스트 처리: wc, sort, comm
- 필터: grep, cut, paste, diff
- 고급 텍스트 처리: sed, awk

명령어: tar

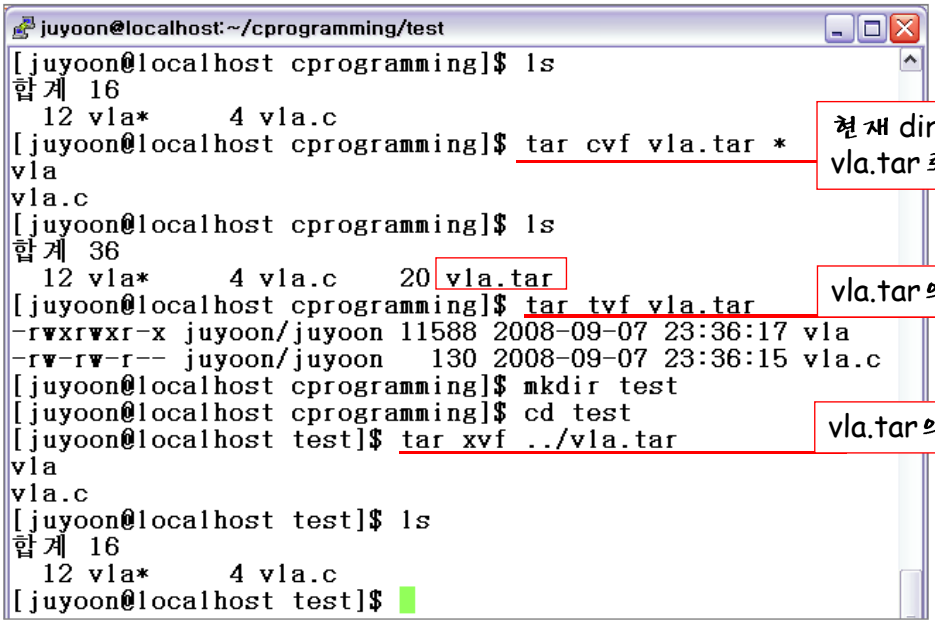
- 저장 또는 이동 목적을 위해 여러 개의 파일을 하나로 묶는다. (tape archive)
  - 묶는 기능만 수행하며 압축하지 않는다.

```
tar [function letters] [tar-file] [options] [file …]
```

- Function Letters
  - c : Create. 묶음 파일을 생성
  - r : Replace. 주어진 파일을 기존 tar 파일 끝에 추가.
  - u : Update. 주어진 파일이 새로운 경우에만 기존 tar 파일 끝에 추가.
  - t : Table of contents. 묶여진 파일들의 목록을 보여준다.
  - x : Extract. 하나의 파일명으로 묶여진 파일을 푼다.
  - f : File. 주어진 파일명으로 tarfile을 만든다.
  - z : Zip. 압축된 tarfile을 만든다. (Linux only)
  - 그 외에도 많음. 상세 기능은 매뉴얼 참조.
  - 기능 문자는 옵션이 아니므로 '–'를 붙이지 않아도 된다.

명령어 : tar

❑ tar 사용 예



- v? - Verbose. 실행 과정 메시지를 출력한다.

명령어 : gzip

❑ 유닉스 환경에서 가장 많이 이용하는 압축 유틸리티 중 하나

```
gzip [options] [name …]
gunzip [options] [name …]
zcat [options] [name …]
```

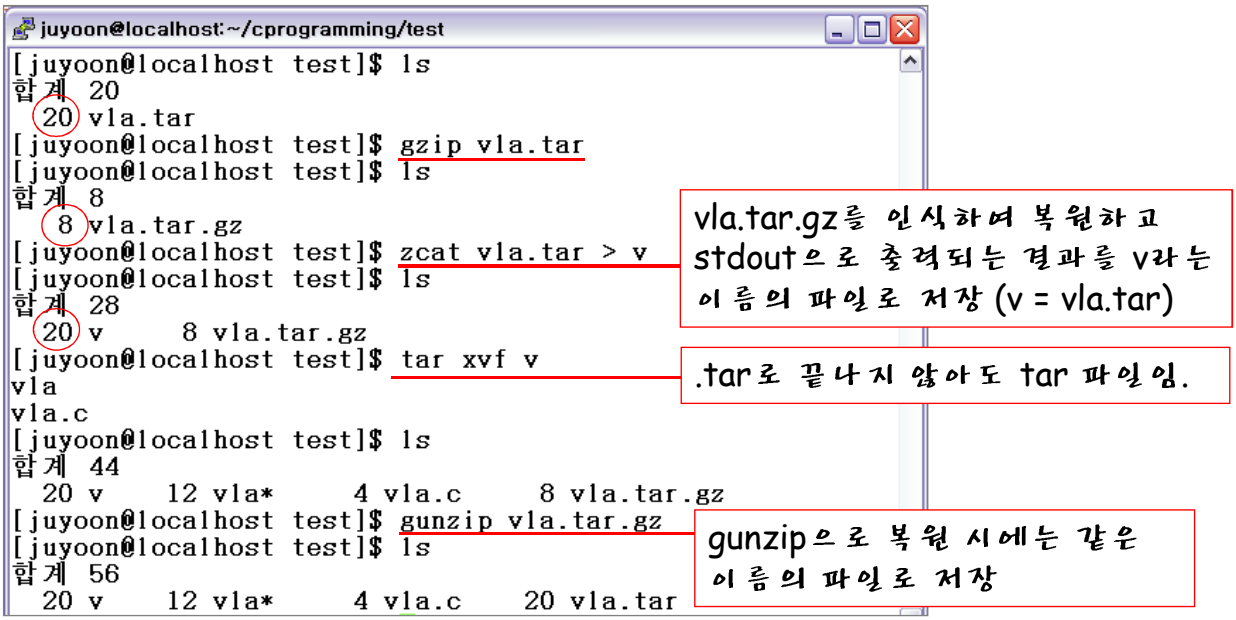
- 압축을 한 후 파일 확장자는 .gz가 붙음
- gunzip = gzip -d
- .Z가 붙은 파일도 복원할 수 있음 (compress/decompress)
- zcat: 압축된 파일을 복원하여 stdout으로 출력

❑ 주요 옵션

- -t: 파일 무결성 체크 (--test)
- -v: 이름과 압축률 표시 (--verbose)
- -h: 도움말 (--help)
- -r: 디렉터리 구조 순환 (--recursive) (* 디렉터리를 압축하지는 않음)

명령어 : gzip

❑ 사용 예



- 복원과 묶음 해제를 동시에
  - "zcat vla.tar | tar xvf -" (-는 stdin을 의미함)

명령어 : bzip2

❑ 최근 많이 사용되는 압축 유틸리티

- gzip보다 압축률이 좋으나 느리다.

```
bzip2 [options] [name …]
bunzip2 [options] [name …]
bzcat [options] [name …]
bzip2recover filename
```

- 압축을 한 후 파일 확장자는 .bz2가 붙음
- bunzip2 = bzip2 -d
- bzcat: 압축된 파일을 복원하여 stdout으로 출력
- bzip2recover: 손상된 압축파일을 복원

❑ 주요 옵션

- -t: 파일 무결성 체크 (--test)
- -v: 이름과 압축률 표시 (--verbose)
- -k: 입력 파일 보존 (--keep)
- -f: 같은 이름의 압축 파일을 덮어 씀 (--force)

압축과 묶음

- 일반적으로 tar와 zip을 함께 사용하여 덩어리를 만든다. → tarball

- Tarball naming convention
  - .tar.gz == .tgz
  - .tar.bz2 == .tbz
  - .tar.Z == .taz
  - .tar.lzma == .tlz
  - gunzip/bunzip2 등의 유틸리티는 새로운 이름도 인식
    - 복원결과는 .tar

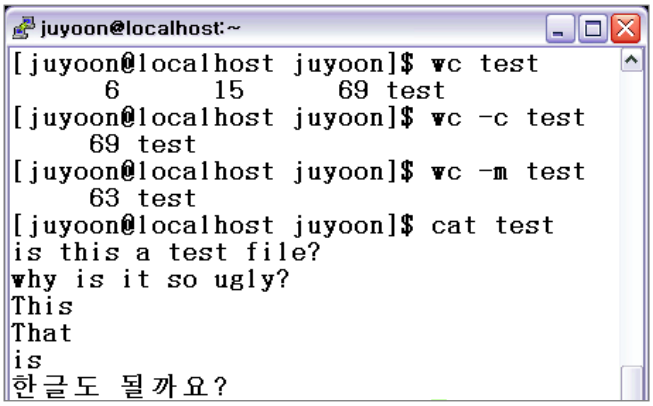
명령어: wc

- 지정한 파일 내에 있는 줄 수, 단어 수, 문자 수에 대한 정보를 출력하는 명령어

```
wc [options] [file …]
```

  - 옵션을 지정하지 않으면 그 파일의 줄 수, 단어 수, 문자 수 및 파일명 모두를 표준 출력함
  - 주요 옵션

옵 션	의 미
-l	줄 수
-w	단어 수
-c	문자 수 - bytes
-m	문자 수 - characters



명령어 : sort

❑ 정렬, 합병(merge), 또는 텍스트 파일의 순서 체크

```
sort [options] [file …]
```

- 인수로 주어진 파일 내용의 전체 줄을 대상으로 ASCII 코드순으로 정렬되어 화면에 출력
- 파일명을 지정하지 않으면 키보드에서 표준입력으로 읽음 (^D로 끝냄)

명령어 : sort

❑ 주요 옵션

옵션	의 미
-b	필드의 비교에 있어서 공백(space와 tab)으로 시작되는 필드 즉, 앞에 붙는 공백을 무시한다.
-c	파일이 정해진 순서대로 정렬되었는가를 확인하기 위해 정렬 안된 상태의 최초 줄을 출력한다.
-d	사전식 순서로 정렬시킨다. 이는 숫자, 문자, blank(space와 tab)만을 비교한다.
-f	파일 내의 소문자를 대문자로 간주하여 정렬시킨다. 즉 대소문자를 구별하지 않는다.
-k 숫자[,숫자]	정렬의 키가 되는 위치를 선택한다. 숫자는 필드(열) 번호로 1부터 시작하며 범위로 줄 수 있다.
-m	이미 정렬된 파일을 합병시킨다.
-n	문자열의 숫자 부분을 산술적 값으로 정렬시킨다. 즉 숫자들을 그 값 순으로 정렬시킨다.
-r	역순으로 정렬한다.
-u	2개 이상의 줄이 중복될 때 1줄만 출력한다.
-o 파일명	지정된 파일명으로 출력하며 출력 파일명을 지정하지 않으면 화면에 표준 출력 한다.
-t 문자	문자를 필드 구분 문자로 취급한다.

명령어 : comm

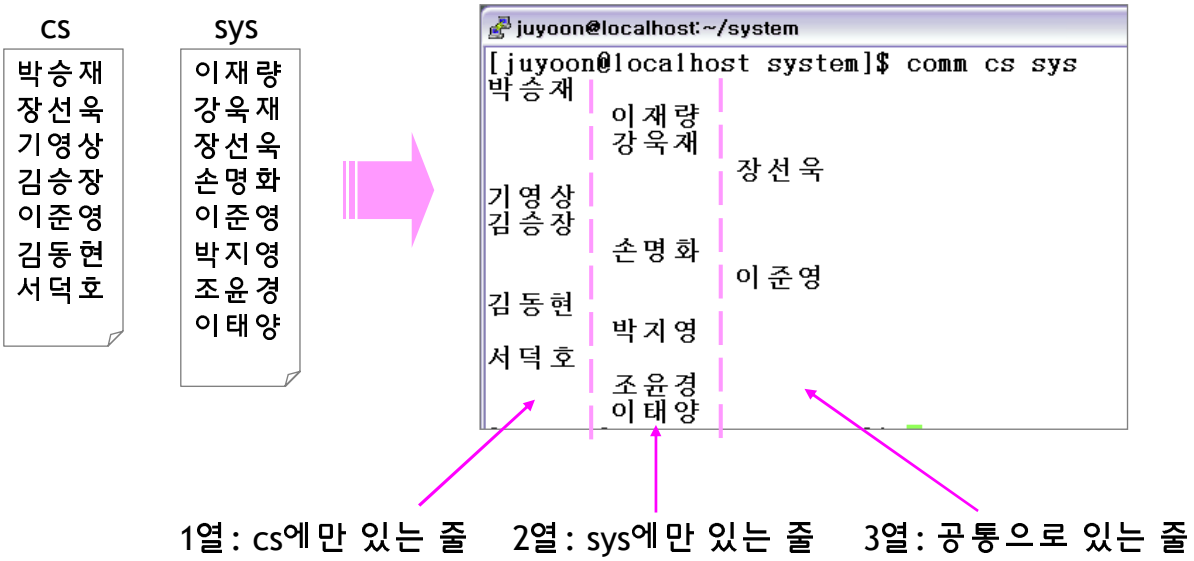
❑ 두 개의 (정렬된) 파일을 줄 단위로 비교하여 공통 부분을 선택 또는 삭제 (common)

```
comm [options] file1 file2
```

- 옵션
  - -1: file1에만 있는 줄 삭제
  - -2: file2에만 있는 줄 삭제
  - -3: 두 파일에 모두 있는 줄 삭제

명령어 : comm

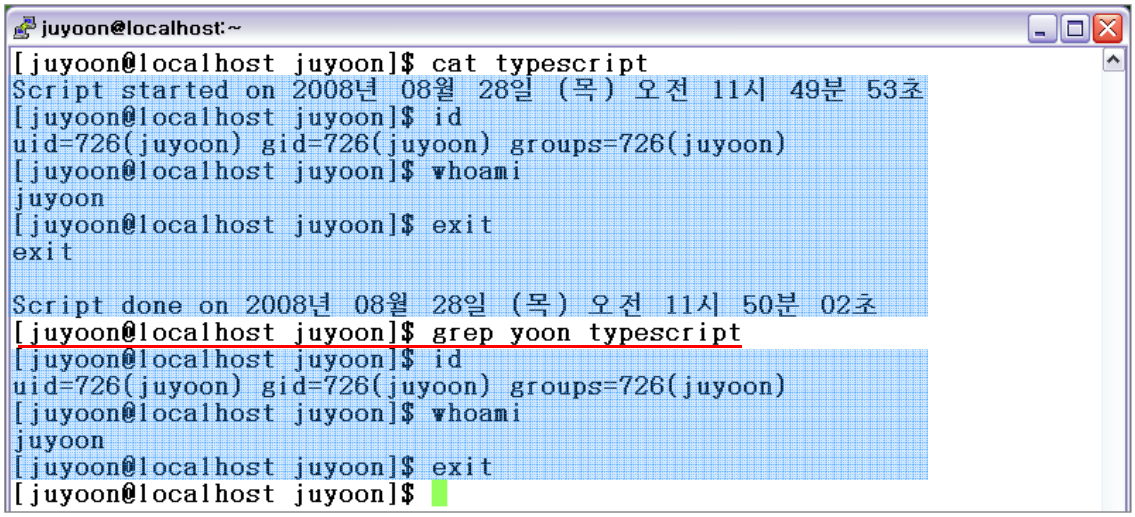
❑ 사용 예



명령어 : grep

❑ 파일에서 지정한 정규식을 만족하는 패턴 찾기 (global regular expression print)

```
grep [options] pattern [file …]
```



명령어 : grep

❑ 옵션

옵션	의미
-b	찾는 각 줄 앞에 블록 번호를 붙여서 출력한다.
-c	찾은 줄 내용은 출력하지 않고 줄 수만 출력한다.
-e pattern	-로 시작하는 pattern 정의
-E	Extended regular expression 사용 (← egrep)
-f file	Pattern이 file에 정의되어 있다. 한 줄에 한 pattern
-F	Pattern을 정규식이 아닌 일반 문자열로 해석 (← fgrep)
-i	비교 시 영문자의 대문자, 소문자를 구별하지 않고 비교한다.
-w	지정한 패턴(문자열) 전체와 일치하는 내용을 출력한다.
-n	찾은 각 줄 앞에 줄 번호를 붙여서 출력한다.
-l	결과를 출력할 때 지정한 패턴이 있는 파일명만 출력한다.
-x	지정된 패턴이 한 라인 전체와 일치하는 내용만을 출력한다.
-r	지정된 디렉토리와 그 서브 디렉토리의 모든 파일을 순환적으로 검색하여 파일 내에서 문자열이 포함된 라인을 화면에 출력한다.
-v	지정한 패턴이 없는 행들만 출력한다.

명령어 : grep

❑ 정규식 (Regular Expression)

- 문자열 검색 시 다양한 형태의 패턴을 표현
  - vi, grep 등에서 사용하며, 쉘 스크립트, awk, Perl 등의 스크립트 언어와 google, naver 등 검색 사이트에서도 사용
  - 정규식 문법은 정해져 있으나 표현 방식은 환경에 따라 조금씩 다르다.

❑ 정규식의 문법

- 일반 문자와 메타 문자의 조합
- 구성 요소
  - 연결 : abc
  - 선택 : a 또는 b
  - 반복 : ?(zero or one), *(zero or more), +(one or more)

명령어 : grep

❑ 정규식의 문법 (계속)

- POSIX 메타문자

메타문자	의미	사용 예
.	하나의 문자 (\n 제외)	a.c: abc, afc 등과 매치
[ ]	안에 있는 문자 중 하나	[a.c]: a 또는 . 또는 c [a-z]: 소문자 중 하나 [a\-z]: a 또는 - 또는 z
[^]	안에 없는 문자 중 하나	[^a-z]: 소문자가 아닌 문자 하나
^	문자열 또는 행의 시작	^[hc]at: 행의 맨 앞이 hat나 cat으로 시작
$	문자열 또는 행의 끝	[hc]at$: 행의 맨 끝이 hat나 cat으로 끝남
\(\)	그룹	

명령어 : grep

❑ 정규식의 문법 (계속)

- POSIX 메타문자

메타문자	의미	사용 예
*	0번 이상 반복 (패턴의 뒤에 붙인다)	ab*: a, ab, abb, abbb, ... [xyz]*: x, y, z, xx, yy, xz, zyxxy, ... \(ab\)*: ab, abab, ababab, ...
{n}	정확히 n번 매치	
{n,}	n번 이상 매치	
{n,m}	n번 이상, m번 이하 매치	

명령어 : grep

❑ 정규식의 문법 (계속)

- POSIX ERE (Extended Regular Expression)

메타문자	의미	사용 예
+	1번 이상 반복	ab+: ab, abb, abbbb, ...
?	0번 또는 1번	[hc]?at: at, hat, cat
|	선택	(mk|rm)dir: mkdir, rmdir

❑ 정규식의 문자집합

[:alnum:]	A-Za-z0-9	[:alpha:]	A-Za-z
[:word:]	A-Za-z0-9_	[:blank:]	\t
[:cntrl:]	\x00-\x1F\x7F	[:digit:]	0-9
[:graph:]	\x21-\x7E	[:lower:]	a-z
[:print:]	\x20-\x7E	[:space:]	\t\r\n\v\f
[:upper:]	A-Z	[:xdigit:]	A-Fa-f0-9
[:punct:]	-!"#$%&'()*+,./:;<=>?@[\\\]_'{|}~		

명령어: cut

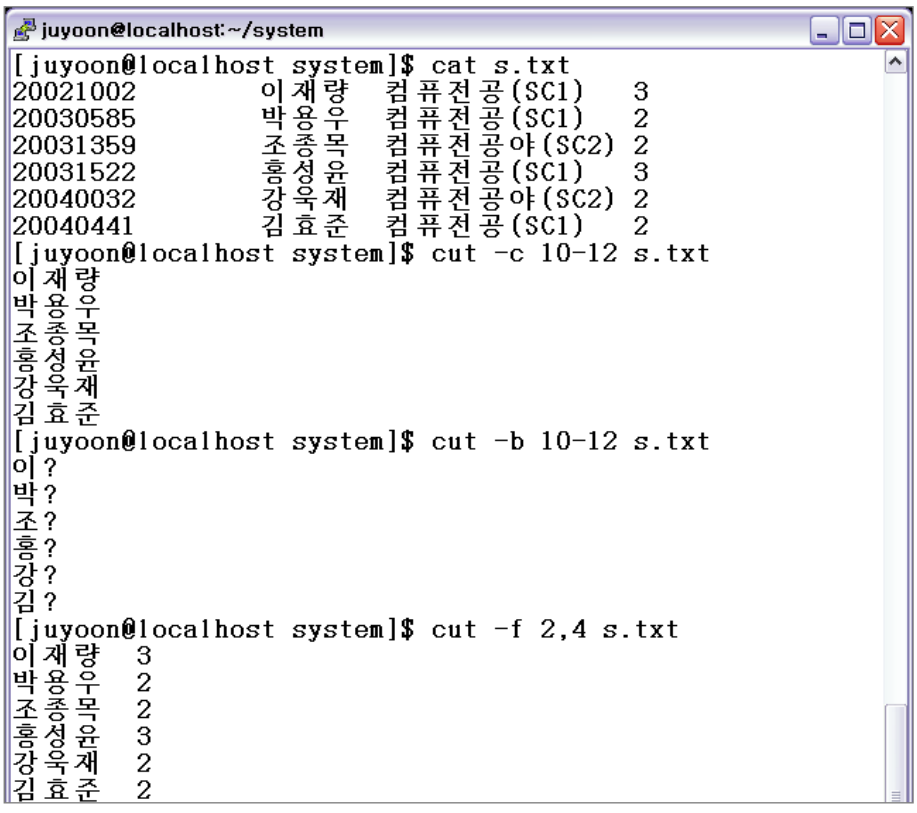
❑ 조건에 따라 데이터의 열을 추출하는데 사용

```
cut –b list [-n] [file …]
cut –c list [file …]
cut –f list [-d delimiter] [-s] [file …]
```

- 각 줄에서 지정한 열이나 범위가 정해진 영역(필드라고 함)을 추출
- 명령 요소
  - list: 추출할 열의 목록(오름차순으로)
    - 범위는 1,4,7; 1-3,8; -5,10 (1-5,10의 의미)와 같이 지정
  - file: 작업 대상 파일
  - -b: byte 단위로 위치 지정
  - -c: 문자 단위로 위치 지정
  - -f: 구분자(delimiter)로 각 필드 구분한 파일 사용. 구분자는 –d 옵션으로 지정하며 default는 tab

명령어: cut

❑ 사용 예

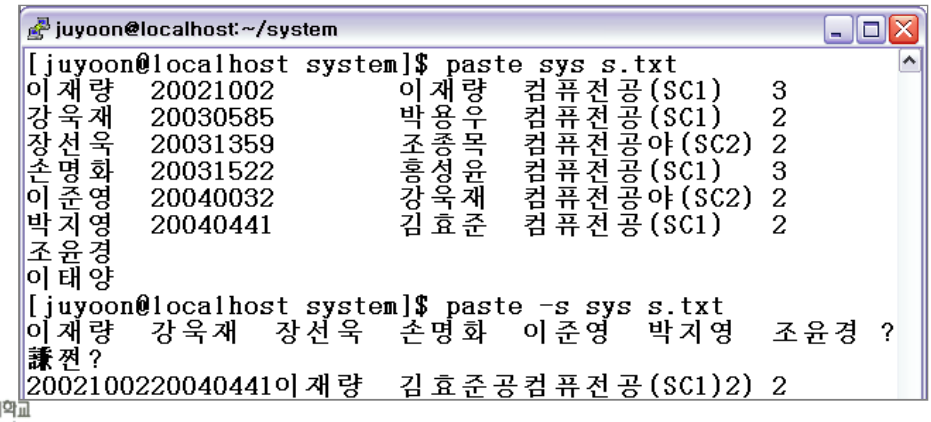


명령어 : paste

❑ 사용자가 지정한 두 개 이상의 파일 내용 중에서 같은 줄을 연결시키거나 한 개의 파일에서 줄을 연결

```
paste [-s] [-d list] [file …]
```

- 옵션
  - -d list: tab 대신 list에 있는 글자들을 구분자로 사용
  - -s: 파일에 있는 모든 줄을 한 줄로 결합 (serial)



명령어 : diff

❑ 두 파일을 줄 단위로 비교하여 차이점을 찾는다.

```
diff [options] file1 file2
```

- 주요 옵션

옵션	의미
-b	연속된 빈 공간 (space, tab 등)을 하나의 같은 것으로 취급한다.
-i	대소문자를 구분하지 않는다.
-w	빈 공간(space, tab)을 무시하고 나머지 글자들만 비교한다.
-c	출력 형태를 파일정보, 각 파일의 차이점 순서로 바꾼다.

명령어 : diff

❑ 사용 예

```
[juyoon@localhost system]$ diff s1.txt s2.txt
1,5c1,6
< 20021002      이재량   컴퓨전공(SC1)   3
< 20030585      박용우   컴퓨전공(SC1)   2
< 20031522      홍성윤   컴퓨전공(SC1)   3
< 20040441      김효준   컴퓨전공(SC1)   2
< 20050515      김민준   컴퓨전공(SC1)   3
---
> 20021002      이재량   컴퓨전공(SC1)          3
> 20030585박용우컴퓨전공(SC1)2
> 20040441      김효준   컴퓨전공(SC1)   2
> 20040549      박보현   컴퓨전공(SC1)   2
> 20041273      정동구   컴퓨전공(SC1)   2
> 20050240      김성     컴퓨전공(SC1)   2
[juyoon@localhost system]$ diff -b s1.txt s2.txt
2,3c2
< 20030585      박용우   컴퓨전공(SC1)   2
< 20031522      홍성윤   컴퓨전공(SC1)   3
---
> 20030585박용우컴퓨전공(SC1)2
5c4,6
< 20050515      김민준   컴퓨전공(SC1)   3
---
> 20040549      박보현   컴퓨전공(SC1)   2
> 20041273      정동구   컴퓨전공(SC1)   2
> 20050240      김성     컴퓨전공(SC1)   2
[juyoon@localhost system]$ diff -w s1.txt s2.txt
3d2
< 20031522      홍성윤   컴퓨전공(SC1)   3
5c4,6
< 20050515      김민준   컴퓨전공(SC1)   3
---
> 20040549      박보현   컴퓨전공(SC1)   2
> 20041273      정동구   컴퓨전공(SC1)   2
> 20050240      김성     컴퓨전공(SC1)   2
[juyoon@localhost system]$
```


명령어 : sed

❑ 파일을 한 줄씩 읽어 편집 (stream editor)

```
sed [options] script [file1 …]
```

- script: 편집 명령

```
[address [, address]] command [arguments]
```

  - sed의 편집 명령은 vi에서 사용할 수 있다.
    - 차이점은?

- 주요 옵션
  - -f scriptfile: scriptfile의 내용대로 적용
  - -i: file의 내용을 즉시 변경

```
[juyoon@localhost system]$ m s3
This is my cat. my cat's name is betty.
This is my dog. my dog's name is frank.
This is my lizard. my lizard's name is liz.
[juyoon@localhost system]$ sed '1,2s/₩. / and /' s3
This is my cat and my cat's name is betty.
This is my dog and my dog's name is frank.
This is my lizard. my lizard's name is liz.
```

명령어: sed

- address
  - *number*: 줄 번호
  - *number1*~*number2*: *number1* 줄부터 시작해서 *number2* 줄마다
  - *addr1*,*addr2*: *addr1* 줄부터 *addr2* 줄까지
  - *addr*,+*number*: *addr*부터 시작해서 *number* 줄만큼
  - $: 마지막 줄
  - /*regexp*/: *regexp*와 매치되는 모든 줄
- command
  - #: comment문 - 줄 끝까지 적용
  - q: sed를 끝낸다.
  - d: 현재 처리 중인 줄을 끝내고 다음 작업 시작
  - p: 현재 처리한 줄을 표준 출력.
  - n: 다음 줄 처리 시작 (pattern space에 줄을 넣음)
  - { … }: 여러 명령을 한 번에 처리하고 싶을 때 { }로 싼다.
  - s: s/*regexp*/*replacement*/*flags* 와 같은 형식으로 사용 (substitute)
    - *regexp*에 해당하는 패턴을 *replacement*로 대체
    - flag: 추가 실행 요소 (g: global, 그 줄의 모든 경우에 적용)

명령어: sed

- 일반적으로 script 파일을 만들어 프로그램화하여 실행시킨다.

```
#!/bin/sed -f
#버퍼에 80칸을 넣는다
1 {
  x
  s/^$/
  s/^.*$/&&&&&&&&/
  x
}
#맨 앞과 맨 뒤의 연속
y/(tab)/ /
s/^ *//
s/ *$//

#줄 끝에 \n과 새로운
G

#처음 81글자(80칸+\
s/^\(.\{81\}\).*$/
#남은 공간의 반을 잘라 앞으로 보낸다.
s/^\(.*\)\n\(.*\)\2/\2\1/
```

```
juyoon@localhost:~/system
[juyoon@localhost system]$ chmod +x s.script
[juyoon@localhost system]$ s.script s2
                              This is my cat
                           my cat's name is betty
                              This is my dog
                           my dog's name is frank
[juyoon@localhost system]$ sed 's/is/are/' s2
Thare is my cat
 my cat's name are betty
Thare is my dog
 my dog's name are frank
[juyoon@localhost system]$ sed 's/is/are/g' s2
Thare are my cat
 my cat's name are betty
Thare are my dog
 my dog's name are frank
```

- 상세 설명서
  - http://www.gnu.org/software/sed/manual/sed.html

명령어: awk

❑ 텍스트 처리를 위한 스크립트 언어

- Aho, Weinberger, Kernighan
- C, Bourne shell ~ Perl, javascript
- 언어이므로 실행환경이 있는 어디서나 사용 가능 - Windows에서도 가능

❑ 사용법

- 쉘 프롬프트에서: `awk 'program' [file …]`
- 별도의 프로그램 파일: `awk -f progfile [file …]`
- 자체 실행
  - 프로그램 맨 앞에 `#!/bin/awk -f`를 적는다.
  - 프로그램 파일의 모드를 실행 가능하도록 설정하고 쉘 프롬프트에서 파일 이름을 호출한다.

명령어: awk

❑ 프로그램 예

b.awk

```
BEGIN {print "File\tOwner"}
      {print $10, "\t", $4}
END {print " - DONE -"}
```