Linux Backspace/Delete mini-HOWTO

Sebastiano Vigna

vigna@acm.org

도운이

정하녕

alita@kldp.org

고친 과정

고침 v1.42000년 12월 7일

Red Hat 7.0와 Helix Gnome의 충돌에 대한 사항 갱신

고침 v1.32000년 10월 18일

이름 변경.

고침 v1.22000년 10월 15일

내용 갱신. "작동하지 않을 때는" 절 추가.

고침 v1.12000년 9월 13일

tcsh 수정사항 추가

고침 v1.02000년 9월 5일

첫 배포

1. 소개

모든 리눅스 사용자들은 **Backspace**와 **Delete** 키를 콘솔과 X 상에서 제대로 작동하도록 하는 것이 불가능해 보이는 상황을 겪어본 적이 있다. 이 글은 이런 일이 왜 일어나는지에 대해 설명하고 해결책을 제시한다. 여기에 주어진 개념들은 배포판마다 다를 수 있으며 각 배포판마다 시스템 설정 파일들이 매우 다르기 때문에 여기서는 여러분이 필요한 경우 자신에게 해당하는 수정 사항을 만들 수 있도록 충분한 정보를 제공하려 한다.

이 문서에서는 **Backspace** 키는 이전으로 한 칸 움직여서 커서가 가리키는 문자를 지워야 한다고 가정한다. 반면 **Delete** 키는 커서의 이동 없이 현재 커서가 가리키는 문자를 지워야 한다. 대부분의 키보드들이 왼쪽을 가리키는 화살표(←)를 **Backspace** 키 위에 그려놓았지만 여러분들이 이 두 키들의 기능을 바꾸려 할 때 이 문서가 직접적인 방법을 제공하지는 않지만 여기 있는 내용들이 도움이 될 것이다.

또한, 수정 사항들은 사용자 파일들에만 적용되도록 했다. 배포판의 표준적인 부분들은 건

드리지 않는다. 최종적으로 이 문서는 어플리케이션들이 정확한 작동을 하도록 시스템을 설정하는 방법에 대해 논의한다.

> 참고**:** 이 Mini-HOWTO가 처음 배포된 이후 많은 것들이 더 복잡해졌다. 동일한 터미날 에뮬레이터조차 서로 다른 배포판에서 다른 ASCII 문자열을 만들어낸다. (예를 들어 Red Hat 7.0이나 7.0 이하 버전의 Helix/Red Hat 에서의 gnome-terminal처럼). 이러한 불일치 때문에 현재의 터미날 데이타베이스는 예정되었던 것보다도 터미날 에뮬레티어와 연관성이 적게 되었다. 이하 논의에 대한 하나의 기준으로 삼기 위해 Debian keyboard guidelines (http://www.debian.org/doc/debian-policy/ch3.html#s3.8)에서 제안되는 설정이 표준적인 것으로 간주하겠다.

2. 키가 작동하는 방법

키보드의 키가 눌려지면 수많은 하드웨어와 소프트웨어 부분들이 작동을 해서 키의 의도된 동작 (예를 들어 특정 문자를 발생시킨다거나 하는) 이 실제로 일어나도록 해준다. 여기서는 소프트웨어 부분에 대해서만 설명할 것이며 (하드웨어 부분에 대해서는 제어가 불가능하다) 일단은 콘솔 출력과 관련된 것에 대해서만 얘기하겠다.

1. 키를 누르면 키보드 *스캔코드*가 발생한다. 이 스캔코드들은 다시 *키코드*로 변환된다. i386 시스템에서는 일반적으로 **Backspace** 키는 키코드 14를, **Delete** 키는 키코드 111을 발생시킨다.
2. 이후 키보드 라이브러리는 사용자가 지정한 키보드 정의에 의해서 키코드를 *키보드 심볼 (키심볼)*로 변환한다. 키보드 데이타베이스를 들여다 본다면 (Red Hat Linux의 경우엔 `/usr/lib/kdb/`) 서로 다른 컴퓨터, 레이아웃, 같은 키에 대한 다른 해석에 따라서 여러 개의 정의가 있는 것을 알 수 있다 (예를 들어 어떤 사용자는 두 **Alt** 키들이 서로 다른 modifier로 작동하길 원한다). 리눅스 콘솔의 키보드 레이아웃은 키심볼 Delete를 키코드 14로, 키심볼 Remove를 키코드 111로 지정하고 있다. 이는 이상하게 보일지도 모르나 리눅스 콘솔은 VT100 터미날처럼 작동하도록 되어있으며 이 터미날에서도 실제로 이렇게 작동한다.
3. 아직 과정이 끝난 것이 아니다. 콘솔 어플리케이션들은 키심볼이 아닌 ASCII 문자열을 읽어들인다. 따라서 콘솔은 키심볼을 읽어 들여서 이를 키를 알맞게 나타내는 ASCII 문자열로 변환해 줘야 한다. 물론 이 작업은 어플리케이션이 알아들을 수 있는 방식으로 행해져야 한다. 예를 들어 리눅스 콘솔상에서 Delete 키심볼은 ASCII 코드 127 (DEL)로, Remove 키심볼은 적당한 문자열로, BackSpace 키심볼은 ASCII 코드 8 (BS)로 변환된다.
4. 마지막으로 다시 앞으로 돌아가서 각 키에 의해 생성된 ASCII 문자열을 *키 기능*으로 변환해 주어야 한다. 이 일은 *터미날 데이타베이스*가 해주는데 이 터미날 데이타베이스는 각각의 터미날 종류에 대해서 문자열에서 키 기능 (기본적으로는 키심볼의 부분집합이다) 으로의 변환 정보를 가지고 있다.

   > 참고**:** 불행히도 터미날 데이타베이스는 termcap과 terminfo의 두 "표준"이 존재한다. 사용하는 배포판에 따라서 둘 중 하나를 사용하거나 각 어플리케이션에 따라서 다를 수도 있다. 우

리의 논의는 좀 더 현대적인 terminfo 데이타베이스에 맞춰 나갈 것이지만 제공되는 수정사항들은 둘 모두를 염두에 둔 것이다.

예를 들어, 리눅스 콘솔상에서 **F1** 키는 `[[A` 이스케잎 문자를 발생시키며 이 값은 터미날 데이타베이스의 콘솔 항목에 따라서 `key_f1` 기능으로 해석될 수 있다 (각 항목을 보고 싶으면 **infocmp linux** 명령을 사용하면 된다). 터미날 데이타베이스에 대한 자세한 내용들은 GNU의 termcap 메뉴얼을 보면 된다. 보통 리눅스 어플리케이션들은 ncurses 패키지에 포함되어 있는 terminfo 데이타베이스를 사용한다.

이 시점에서는 크게 놀라운 일도 아니겠지만, 리눅스 콘솔의 terminfo 항목은 DEL을 `kbs`(backspace 키) 기능으로, `[3~` 이스케잎 문자를 `kdch1` (“한 문자 지움” 키) 기능으로 변환한다. 처음에 **Backspace** 키가 DEL을 발생시키는 것이 이상했지만 터미날 데이타베이스가 모든 것을 제자리로 돌려놓으므로 정상적으로 작동하는 어플리케이션들은 DEL을 `kbs`로 해석하여 커서 왼쪽의 한 문자를 지우게 된다.

3. 왜 (항상) 작동하지 않는가

이제 기본적인 문제가 명확하리라 본다. 문제는 키보드와 콘솔 어플리케이션들 사이에서 존재하는데 이 둘이 오직 ASCII 문자열만을 통해서 통신이 가능하다는 사실이다. 따라서 특수키들은 키심볼에서 문자열로 변환되고 다시 이 문자열이 키 기능으로 바뀌어져야 한다. 다른 콘솔들은 이 변환에 대해 서로 다른 기준을 가지고 있기 때문에 우리는 터미날 데이타베이스가 필요한 것이다. 아주 작은 문제를 제외하곤 시스템은 제대로 작동할 것이다. 문제는, 모든 시스템이 제대로 설정된 것은 아니며 모두가 터미날 데이타베이스를 사용하는 것은 아니라는 점이다.

어플리케이션은 사용할 데이타베이스 항목을 알 수 있는 방법이 필요한데 이는 보통 TERM 환경변수를 알맞게 설정하면 된다. 일부 경우엔 TERM이 지정하는 데이타베이스 항목과 터미날 에뮬레이터간의 불일치가 있기도 하다.

더군다나 많은 어플리케이션들은 터미날 데이타베이스를 (적어도 전부를) 사용하지 않고 BS와 DEL ASCII 코드들을 의도한 의미로 해석한다. 다시 말해 데이타베이스를 찾아보지도 않고 각 코드들에 기능을 할당한다 (물론 일반적으로 그 기능들은 커서가 가리키는 혹은 커서 이전의 문자를 지우는 것이다). 그 결과 이제 우리의 멋진 체계는 완전히 망가져 버렸다 (모든 리눅스 사용자들이 뼈저리게 알고 있는것 처럼). 예를 들어 bash는 DEL이 앞쪽 한 문자 지움을, 다시 말해 backspace 기능을 해야 한다고 가정한다. 결국 일반적인 설치 환경에서 **Backspace** 키는 콘솔 상에서 예상대로 작동하는데 이는 두 번 꼬아진 이유 때문이다! 물론 **Delete** 키는 작동하지 않는다. 이는 bash가 `kdch1` 기능에 대해서 터미날 데이타베이스를 찾아보지 않기 때문이다.

일들이 어떻게 꼬아져 있는지를 보기 위해서 Red Hat 배포판 (그리고 여타 다른 배포판들도 아마) 에서 제공되는 **fix_bs_and_del** 스크립트를 보자. 이 스크립트는 몰래 BackSpace 키심볼을 **Backspace** 키로, Delete 키심볼을 **Delete** 키로 할당해준다. 이제 쉘은 제대로 작동한다! 그러나 키심볼과 터미날 데이타베이스의 정상적인 관계에 의존하는 프로그램들은 모두 정상작동을 하지 않게 된다. 이는 Delete 키심볼이 DEL로 변환되고 이것이 다시 terminfo 데이타베이

스에 의해 `kbs` 키 기능으로 변환되어 그런 - 정상적인 - 프로그램들에서는 두 키가 모두 backspace 기능을 갖게 되기 때문이다.

4. X

X 하에서의 상황도 크게 다르지는 않다. 단지 다른 계층이 있어서 X 윈도우 시스템이 스캔코드를 자신만의 키 심볼로 변환하며 이 값들이 콘솔의 그것들 보다 더 다양하고 정확하다. 그 후 이 값들을 어플리케이션으로 넘겨준다 (이런 이유로 Xemacs이 앞의 문제를 겪지 않을 수 있다. X는 키코드 14를 키심볼 BackSpace으로, 키코드 111을 키심볼 Delete으로 변환하며 사용자는 이 키심볼들에 원하는 기능을 쉽게 부여할 수 있다). 물론, 터미날 에뮬레이터 프로그램 (흔히 X 상에서의 VT100 에뮬레이터) 은 X의 키심볼들을 ASCII 문자열로 변환해야 하며 우리는 다시 이전과 같은 문제에 봉착한다.

더 자세히 말하면, 보통 xterm은 정확히 콘솔과 똑같이 작동한다 (다시 말해 똑같은 ASCII 문자열을 생성한다). 그러나 7.0 미만 버전의 Red Hat의 gnome-terminal 같은 경우는 **Backspace** 키에 대해선 BS를, **Delete** 키에 대해선 DEL을 발생시킨다. 재밌는 사실은 기본적으로 이 두 어플리케이션들이 동일한 터미날 데이타베이스 항목을 사용한다는 것이며 따라서 `kbs` 기능이 ASCII DEL로 연결된다는 사실은 정상적으로 작동하는 모든 어플리케이션들을 **Backspace**와 **Delete** 키들에 대해서 gnome-terminal과 동일하게 동작하도록 만든다.

```
bash$ export TERM=gnome
```

위와 같은 갇단한 문장이 정상적일 어플리케이션들의 경우에는 이 문제를 해결해 줄 수 있다. 그러나 여러분 시스템의 터미날 데이타베이스에 `gnome`이라는 항목이 없을 수 있으므로 - 특히 최신의 데이타베이스가 아닌 경우에 - 항상 적용되는 것은 아니다.

어떤 경우에는 이 방법은 해결책이 아닐 수 있다. 예를 들어 버전 7.0 이상의 Red Hat 배포판을 사용한다면 gnome-terminal은 콘솔처럼 작동한다. 그러나 Helix 배포판을 사용하여 데스크탑을 업그레이드 했다면 gnome-terminal은 7.0 이전 Red Hat의 경우처럼 작동하게 된다.

앞으로의 논의를 간단히 하기 위해 우리는 표준을 콘솔처럼 작동하는 VT100 에뮬레이터로, 변종을 **Backspace** 키에 대해서 BS를 **Delete** 키에 대해선 DEL을 발생시키는 것으로 정의할 것이다. 따라서 예를 들어 xterm은 데비안 배포판에서는 이전부터 계속 표준이었지만 Red Hat에서는 표준에서 변종으로, 혹은 그 반대로 몇 번 바뀌었다. gnome-terminal의 동작은 더욱 많이 변했다.

5. 어플리케이션을 만들 때 해줘야할 일

콘솔 어플리케이션을 만든다면 사용자에게 친절하도록 해야 하고 아래의 fallback 흐름을 이용하여 표준 입력에서 들어오는 것을 이해하도록 해야 한다.

1. 알맞은 terminfo 항목을 열어서 문자열이 현재의 터미날에서 특별한 의미를 갖는지를 찾아본다. 만약 그렇다면 terminfo의 의미를 사용한다.

2. 줄바꿈 문자와 새줄 문자, 탭, BS, DEL 문자들에 대해서 ASCII에서 의도하는 의미를 사용한다. 행운을 빌면 잘 될지도 모른다. ;=)

6. 시스템에서 해줘야할 일

시스템을 표준에 맞게 하고자 하는 사람들을 혼란스럽게 만드는 주된 문제는 그들이 종종 잘못된 방식으로 일을 한다는 것임을 상기하자. 정상적으로 작동하는 부분들은 흔히 우연히 그렇게 작동하는 것이기 때문에 어떤 것이 잘못되었다는 가정 하에 시스템을 수정하려 하는 것은 정상적인 설정까지도 틀리게 만들기도 한다.

6.1. 해야할 일

6.1.1. 변종을 찾는다

깔끔한 해결책을 위한 첫 걸음은 어떤 터미날이 변종이고 어떤 것이 아닌지를 정확히 아는 것이다. 흔히 터미날들은 모두 콘솔처럼 작동하며 이런 경우에 매우 적은 양의 수정만으로 모두 정상적으로 작동하도록 할 수 있다. 그러나 변종 터미날 (gnome-terminal의 변종같은) 을 사용한다면 이를 특별한 방법으로 다뤄줘야 한다.

아래의 한 줄짜리 C 프로그램

```
void main(void) {int c; while(c = getchar()) printf("%d 0x%02X\n", c, c);}
```

이 도움이 될 수도 있다. 이 프로그램을 `ascii.c`이라는 이름으로 저장한 후 **gcc ascii.c -o ascii** 명령으로 컴파일하고 **./ascii**로 실행시킨다. 그리곤 아무 키나 누른 후 **RETURN**을 누른다. 이 프로그램은 발생된 ASCII 문자열의 코드들을 10진수와 16진수 형태로 보여준다 (모든 코드들을 보길 원한다면 우선 **stty erase ^-** 명령을 실행한다). 이제 **Backspace** 키가 무슨 일을 하는지를 쉽게 볼 수 있다. 이 키가 DEL(127)을 발생시킨다면 표준 에뮬레이터를 사용하는 것이고 BS(8)을 발생시킨다면 변종 에뮬레이터를 사용하고 있는 것이다.

6.1.2. 에뮬레이터간의 구분

변종 에뮬레이터를 사용한다면 이를 표준의 것과 구분해야 한다. 이론적으론 이것은 문제가 되어서는 안되는데 이는 다른 방식으로 작동하는 터미날들에 대해서는 터미날 데이타베이스의 서로 다른 항목이 적용되기 때문이다 (사용되는 항목은 TERM 변수의 값에 따라 다르다).

여기서는 모든 변종 VT100 에뮬레이터들에는 `gnome` 항목이, 표준 에뮬레이터들에는 `xterm` 항목이 적용되도록 하겠다. 이는 일부 배포판들에 잘 적용된다 (예외는 5.0 이하의 RedHat 같은 일부의 경우인데 이런 배포판들에서는 `xterm` 항목도 변종이다).

그러나 gnome-terminal은 기본적으로 xterm과 동일한 항목을 사용하므로 둘 중 하나는 변종이고 다른 하나는 표준이라면 이를 구분할 방법을 찾아야 한다. 이론적으로는

gnome-terminal의 `termname` 옵션이 사용자에게 TERM 변수를 좀 더 적절한 값으로 설정할 수 있도록 해줘야 하지만 gnome-terminal 1.2.1과 같은 경우에는 이 옵션이 작동하지 않는다.

여기서 사용할 수 있는 괜찮은 방법은 gnome-terminal이 COLORTERM 변수를 `gnome-terminal`로 설정한다는 사실이다. 따라서 쉘 설정 파일에 간단한 검사 항목을 추가함으로써 우리는 TERM 변수값을 수정할 수 있게 된다.

6.1.3. 터미날 데이타베이스의 정정

이제 문제는 터미날 데이타베이스가 변종 터미날들에 사용될 `gnome` 항목이 없을 수도 있다는 것이다 (이는 많은 termcap과 terminfo에서 발생한다). 최신의 terminfo 데이타베이스는 `gnome` 항목을 가지고 있지만 말이다. 그러나 gnome-terminal은 특정한 두 키들만 제외하면 xterm과 기본적으로 동일하게 작동하므로 이 항목을 새로 만드는 것도 가능하다.

6.1.4. 쉘 작동방식 정정

The readline library used by the bash and by many other programs to read the input line can be customized so to recognize specific sequences of characters. The customization can also depend on the TERM variable, so once we can distinguish terminals we can do fine tuning of the keyboard.

bash과 많은 프로그램들이 입력줄을 읽어들이기 위해 사용하는 readline 라이브러리는 특정 문자열을 인식하도록 설정이 가능하다. 이러한 설정 또한 TERM 변수값에 의존하게 되며 따라서 일단 터미날들을 구분할 수 있게 되면 키보드를 보다 정교히 조작하는 것이 가능해진다.

더불어 줄 입력을 직접 처리하는 less나 다른 어플리케이션들 역시 올바르게 작동하게 하려면 변종 터미날 에뮬레이터 하에서 삭제 문자가 DEL이 아닌 BS이라는 것을 쉘에 주지시켜야 한다 (표준의 경우에는 **Backspace** 키가 이미 DEL을 생성하므로 우리가 직접 해줄 일은 없다). 이는 **stty** 명령을 사용하면 된다.

6.2. 하는 방법

> **경고**
>
> 이 수정사항들은 일부 문제점을 가지고있다. 첫째로, 특정 터미날들에서만 작동한다. 둘때로, 이론적으로는 (그러나 거의 일어나지 않을 것이다) 다른 터미날들에서의 readline 라이브러리의 정상 작동을 힙들게 할 수 있다. 그러나 이런 두 한계점들은 거의 해롭진 않다.

먼저 **infocmp gnome** 명령으로 현재의 terminfo 데이타베이스 안에 `gnome` 항목이 있는지를 검사한다 (termcap에 대해서는 나중에 다루겠다). 만약 이 항목이 없다면 아래의 명령으로 `~/.terminfo`에 적당한 항목을 만든다.

```
bash$ tic <(infocmp xterm |\
```

```
        sed 's/xterm|/gnome|/' |\
        sed 's/kbs=\\177,/kbs=^H,/' |\
        sed 's/kdch1=\\E\[3~,/kdch1=\\177,/')
```

똑같은 명령을 root가 실행시키면 시스템 전체의 데이타베이스에 항목을 추가하게 된다 (이는 TERMINFO를 `~/.terminfo`으로 설정함으로써 바꿔줄 수 있다). 여러분이 가지고 있는 `xterm` 항목이 이미 변종이라면 (예를 들어 5.0 이전의 Red Hat의 경우) 이 스크립트는 이를 수정 없이 그대로 복사할 것이고 이것은 정확히 원하던 결과이다.

이제 아래 줄들을 `~/.inputrc`[1]에 추가한다:

```
"\e[3~": delete-char
```

이 줄은 readline 라이브러리에게 표준 터미날에서 **Delete** 키를 다루는 표준적인 방식을 알려준다. 그리고 운 좋게도 이것은 다른 터미날들을 간섭하진 않는다. 이와 함께 DEL 문자의 변종 터미날 상에서의 의미에 대해서도 라이브러리에게 알려줘야 하며 다음과 같은 줄을 `~/.inputrc`에 추가하여 할 수 있다.

```
$if term=gnome
DEL: delete-char
$endif
```

xterm도 변종인 경우엔 추가로 세 줄 정도를 더 넣어줘야 한다. 반면에 변종 터미날이 없다면 이 부분은 필요가 없다. 여기서 다루는 모든 수정사항들은 `/etc/inputrc` 파일을 직접 수정하면 시스템 전체에 영향을 주게 된다.

조건문에 의해 변종 터미날 에뮬레이터가 **TERM** 변수가 올바르에 설정되었다는 가정 하에 작동하게 된다는 것을 주목할 필요가 있다. 이를 보장해 주기 위해서 쓸 수 있는 트릭이 몇 개가 있다. 먼저, gnome-terminal에 기본으로 사용되는 TERM의 값이 `xterm`이기 때문에 모든 터미날이 변종이 아니라면 아무 것도 할 필요가 없다. 그러나 기본으로 `xterm` 항목을 사용하는 터미날이 변종이라면 우리는 TERM 변수를 올바르게 설정해 주는 방법을 찾아야 한다. gnome-terminal의 경우가 그렇다고 가정하자.

이런 효과를 위한 가장 간단한 방법은 gnome-terminal을 `--termname=gnome` 인자를 줘서 실행하는 것이다. 그러나 예전 버전을 가지고 있어서 이 방법이 효과가 없다면 다음 줄들을

```
if [ "$COLORTERM" = "gnome-terminal" ]
then
    export TERM=gnome
fi
```

`~/.bashrc` 설정 파일에 넣어준다[2]. 이 설정문은 gnome-terminal에서만 실행되어서 TERM 변수를 알맞게 설정해준다.

> 참고: 터미날을 `gnome`으로 설정하는면 **ls**에서 색깔을 사용하지 못할 수도 있다. 이는 **ls**의 많은 버전들이 gnome-terminal에서 색깔 사용이 가능하다는 것을 모르기 때문이다. 이를 방지하기 위해선 **dircolors --print-database > ~/.dircolors** 명령으로 `~/.dircolors` 파일을 만근 후 이 파일에 **TERM=gnome** 줄을 추가한다.

이제 변종 터미날 에뮬레이터에 알맞은 termcap 항목을 만들 것이다. 아래처럼 하면 되며 이를 ~/.bashrc에 넣어놓아도 된다.

```
if [ "$TERM" = "gnome" ]
then
    export TERMCAP=$(infocmp -C gnome | grep -v '^#' | \
                    tr '\n\t' '  ' | sed 's/\\  //g' | sed s/::/:/g)
fi
```

마지막으로 터미날 장치에게 삭제 키에 의해서 어떤 문자가 생성되는지를 알려줘야 한다. 일반적으로 삭제 키는 backspace를 의미하므로 우선은 이처럼 설정한 후에 각각의 변종 터미날에 대한 조건문을 추가할 것이다. 물론 ~/.bashrc 안에 넣게된다.

```
stty erase ^?

if [ "$TERM" = "gnome" ]
then
    stty erase ^H
fi
```

여러분이 사용하는 터미날 에뮬레이터가 모두 변종이라면 아무 조건문 없이 삭제 문자를 설정하면 된다.

> 참고**:** 어떤 배포판들은 이 문제들을 전체 시스템용 설정파일인 /etc/inputrc 안에서 교정해 놓고 있다. 이런 경우는 여러분의 ~/.inputrc에서 중복되는 줄들을 빼도 된다.

6.3. tcsh용 수정사항

tcsh의 경우에는 모든 수정사항들이 ~/.tcshrc 안에 들어며 bash의 경우와 동일한 작동을 하게 된다.

```
bindkey "^[[3~" delete-char

if ($?COLORTERM) then
   if ($COLORTERM == "gnome-terminal") then
      setenv TERM gnome
   endif
endif

stty erase ^?

if ($?TERM) then
   if ($TERM == "gnome") then
      setenv TERMCAP \
       "`infocmp -C gnome | grep -v '^#' | tr '\n\t' '  ' | sed 's/\\  //g' | sed s/::/:/g`"
      bindkey "\177" delete-char
      stty erase ^H
   endif
endif
```

두 번째 부분은 모든 변종 터미날에 대해서 똑같이 복사되어야 한다. 물론 termcap 항목이 이미 있다면 새로 만들 필요가 없다.

7. 아무 것도 제대로 돌아가지 않는다면

우선 해야할 일은 앞에서 제시한 한 줄의 C 프로그램을 이용하여 어떤 키가 어떤 ASCII 문자열을 발생시키는지를 이해하는 것이다.

발생되는 문자열을 알고 나면 **infocmp** 명령으로 현재의 terminfo 항목을 본 후 (출력되는 정보가 너무 많더라도 걱정하진 말자!) `kbs`와 `kdch1` 기능들이 알맞은 문자열과 연관되어 있는지를 확인한다 (다시 말해서, 예상하는 키가 발생시키는 문자열). 또한 **stty -a** 명령으로 삭제 문자가 **Backspace** 키가 발생시키는 것인지도 확인한다 (`^H`은 BS을 나타내고 `^?`은 DEL을 나타낸다는 것을 명심한다).

어떤 불일치가 있다면 다음과 같은 몇 가지 이유가 있을 수 있다. TERM 변수 값이 잘못되었거나, 터미날 데이타베이스의 항목이 잘못되었거나 아니면 X 하의 터미날 에뮬레이터가 잘못되었을 수 있다. 이 시점에서 여러분들이 스스로 해결 방법을 찾을 수 있을 만큼 충분한 지식을 가졌길 바란다.

> 참고**:** 다른 어플리케이션들이 서로 다르게 작동한다면 이는 그 중 일부가 터미날 데이타베이스 항목을 잘못 사용하고 있기 때문이다. 어떤 어플리케이션에서 키가 정확한 기능을 하는 것이 그 어플리케이션이 터미날 데이타베이스를 제대로 사용하고 있다는 것을 말해주지는 않는다는 것을 명심한다.—그저 우연히 그렇게 되었을 수도 있다. 개별적으로 검사를 해보고 싶다면 **ne** (http://ne.dsi.unimi.it/) 에디터가 제대로 작동하는지를 본다. **ne**는 `kbs`와 `kdch1`을 포함한 모든 터미날 기능들을 사용하며 자원이 의도한 대로만 작동한다.

8. 결론

여기에서 제시되는 수정은 크게 봐서는 쓴 문자들을 지우는데 있어서의 문제들을 해결해 준다 (그러나 문자들을 쓰는 데에는 도움을 주지 못한다 **:)**).

전체적인 설정에 작은 버그가 하나 있다. gnome-terminal에서 xterm을 실행하면 TERM이 `gnome`으로 설정된다. 물론 이런 불편 사항은 문제가 되지는 않으며 gnome-terminal을 TERM을 알맞게 설정해서 실행시킬 수 있게 되는 즉시 해결된다.

또 하나의 중요치 않은 문제점은 원격지 접속에 관한 것이다. 여러분의 것과 다른 터미날 데이타베이스를 가진 호스트로 접속하는 경우엔 모든 것들을 손으로 설정해 줘야 한다.

마지막으로 이 수정사항들은 잘못 짜여진 어플리케이션들에는 적용되지 않는다는 것을 명심한다 (예를 들어서 `kbs` 기능을 무시하는 어플리케이션). 이런 경우에 해줄 수 있는 것이 거의 없다. 왜냐하면 이런 어플리케이션 하나를 위한 수정이 다른 정상적인 어플리케이션들을 잘못 작동하도록 만들 수 있기 때문이다.

주석

1. 이전 버전의 bash에서는 INPUTRC를 알맞게 설정해 주는 것을 잊어선 안된다. 예를 들어

   ```
   export INPUTRC=~/.inputrc
   ```

   를 `~/.profile` (혹은 로긴 쉘이 읽어들이는 여타 다른 파일) 안에 넣어준다.
2. 더 정확히 말해서 모든 로긴 쉘 뿐 아니라 모든 쉘이 읽어들이는 설정 파일이다.