Linux Backspace/Delete mini–HOWTO

Sebastiano Vigna

vigna@acm.org

Revision History

Revision v1.6 19 Jan 2002

Included many comments from Alex Boldt and Chung–Rui Kao.

Revision v1.5 3 May 2000

Updated for new distros and the tput trick.

Revision v1.4 7 December 2000

Updated for Red Hat 7.0 and Helix Gnome conflicts.

Revision v1.3 18 October 2000

Name change.

Revision v1.2 15 October 2000

Updated. Added "What If Nothing Works" section.

Revision v1.1 13 September 2000

Added tcsh fixes

Revision v1.0 5 September 2000

First release

Table of Contents

1. Introduction....1

2. How Keys Are Turned Into Actions....2

3. Why It Doesn't (Always) Work....3

4. X....4

5. What You Should Do When Writing Applications....5

6. What You Should Do On Your System....6
6.1. What Needs to Be Done....6
6.1.1. Detecting Deviance....6
6.1.2. Distinguishing Between Emulators....6
6.1.3. Fixing the Terminal Database....7
6.1.4. Fixing the Shell Behaviour....7
6.2. How to Do It....7
6.3. Fixing for tcsh....9
Notes....10

7. What If Nothing Works....11

8. More Hacking....12

9. Conclusions....12

1. Introduction

Every Linux user has been sooner or later trapped in a situation in which having working **Backspace** and **Delete** keys on the console and on X seemed impossible. This paper explains why this happens and suggests solutions. The notions given here are essentially distribution−independent: due to the widely different content of system configuration files in each distribution, I will try to give the reader enough knowledge to think up his/her own fixes, if necessary.

I assume that the **Backspace** key should go back one character and then erase the character under the cursor. On the other hand, the **Delete** key should delete the character under the cursor, without moving it. If you think that the function of the two keys should be exchanged, in spite of the fact that most keyboards feature an arrow pointing to the *left* ( ) on the **Backspace** key, then this paper will not give you immediate solutions, but certainly you may find the explanations given here useful.

Another assumption is that the fixes should alter only local (user) files. No standard part of the distribution should be altered. Finally, this document discusses how to set up your system so that applications get the right events. If an application decides to interpret such events in an idiosyncratic way, the only possible fix is to reconfigure the application.

Since the first release of this Mini−HOWTO things have become even more entangled. Different distributions of the same terminal emulator (e.g., gnome−terminal as provided by Red Hat 7.0, Helix Code/Ximian or even Red Hate7.1) generate different ASCII sequences. Due to this mismatch, now the terminal databases correspond even less to the terminal emulators they are supposed to describe. To set a firm ground for the following discussion, we assume basically as correct settings the ones proposed in the Debian keyboard policy.

2. How Keys Are Turned Into Actions

When a key is pressed on the keyboard, a number of hardware and software components cooperate so as to guarantee that the intended meaning of the key (e.g., emitting a certain character) matches the actual behaviour of the key. I will concentrate on the software side (as our control on the hardware part is nonexistent), and in particular, for the time being, on the events related to console output.

1. Hitting a key causes raw keyboard *scancodes* to be generated; these scancodes are then transformed in a *keycode*. On an i386 system, usually the key **Backspace** emits 14 and the key **Delete** emits 111.
2. The keycodes are translated by the keyboard library into a *keyboard symbol (keysym)* using the keyboard definition loaded by the user. If you look into your keyboard database (e.g., in `/lib/kbd/`), you'll discover several definitions for different computers, different layouts and possibly different interpretations of the same keys (e.g., one could desire that the two **Alt** keys really behave as distinct modifiers). The Linux console keyboard layout assigns keysym Delete to keycode 14 and keysym Remove to keycode 111. This may seem strange, but the Linux console emulates a VT100 terminal, and this is the way things work in that realm.[1]
3. Our journey has still to come to an end. Console applications read ASCII sequences, not keysyms. So the console must read keysyms and translate them into ASCII sequences that suitably encode the keys. Of course, this operation must be performed in a way that is understandable by applications. For instance, on the Linux console the Delete keysym is mapped to the ASCII code 127 (DEL), the Remove keysym on a suitable escape sequence, and the BackSpace keysym to ASCII code 8 (BS).
4. Finally, we must in a sense roll back to what we had before and translate the ASCII sequences generated by each key into a *key capability*. This goal is reached by a *terminal database*, which contains, for each kind of terminal, a reverse mapping from sequences of characters to key capabilities (which are essentially a subset of the keysyms).[2]

Unfortunately, there are two "standard" terminal databases, termcap and terminfo. Depending on your distribution, you could be using either one of them, or the database could even depend on the application. Our discussion will concentrate on the more modern terminfo database, but the suggested fixes take both into consideration.

For instance, on the Linux console **F1** generates an escape followed by `[[A`, which can be translated to the capability `key_f1` by looking into the terminal−database entry of the console (try **infocmp linux** if you want to have a look at the entry). A very good and thorough discussion of terminal databases can be found in GNU's termcap manual. Usually Linux applications use the newer terminfo database, contained in the ncurses package.

Maybe at this point not surprisingly, the Linux console terminfo entry maps DEL to the `kbs` (backspace key) capability, and escape followed by `[3~` to the `kdch1` ("delete−one−char" key) capability. Even if you could find strange that the **Backspace** key emits a DEL, the terminal database puts everything back into its right place, and correctly behaving applications will interpret DEL as the capability `kbs`, thus deleting the character to the left of the cursor.

3. Why It Doesn't (Always) Work

I hope the basic problem is clear at this point: there is a bottleneck between the keyboard and console applications, that is, the fact that they can only communicate by ASCII sequences. So special keys must be first translated from keysyms to sequences, and then from sequences to key capabilities. Since different consoles have different ideas about what this translation can look like, we need a terminal database. The system would work flawlessly, except for a small problem: it is not always set up correctly, and not everyone uses it.

Applications must have a way to know which database entry to use: this is accomplished by suitably setting the `TERM` environment variable. In some cases, there is a mismatch between the terminal emulator and the content of the database entry suggested by `TERM`.

Moreover, many applications *do not use* the terminal database (or at least not all of it), and consider BS and DEL ASCII codes with an intended meaning: thus, without looking at the database, they assign them semantics (usually, of course, the semantics is removing the character before or under the cursor). So now our beautiful scheme is completely broken (as every Linux user is bitterly aware). For instance, the bash assumes that DEL should do a backward−delete−char, that is, backspace. Hence, on a fresh install the **Backspace** key works on the console as expected, but just because of two twists in a row! Of course, the **Delete** key does not work. This happens because the bash does not look into the terminal database for the `kdch1` capability.

Just to illustrate how things have become entangled, consider the **fix_bs_and_del** script provided with the Red Hat distribution (and maybe others). It assigns on−the−fly the BackSpace keysym to the **Backspace** key, and the Delete keysym to the **Delete** key. Now the shell works! Unfortunately, all programs relying on the correct coupling of keysym generation and terminal database mappings are now not working at all, as the Delete keysym is mapped to DEL, and the latter to the `kbs` key capability by the terminfo database, so in such programs both keys produce backspacing.

---

4. X

The situation under X is not really different. There is just a different layer, that is, the X window system translates the scancodes into its own keysyms, which are much more varied and precise than the console ones, and feeds them into applications (by the way, this is the reason why XEmacs is not plagued by the problem: X translates keycode 22 to keysym BackSpace and keycode 107 to keysym Delete, and then the user can easily assign to those keysyms the desired behaviour). Of course, a terminal emulator program (usually a VT100 emulator in the X world) must translate the X keysyms into ASCII sequences, so we are again in our sore business.

More in detail, usually xterm behaves exactly like the console (i.e., it emits the same ASCII sequences), but, for instance, gnome−terminal in Red Hat <7.0 or e7.1 emits BS for **Backspace** and DEL for **Delete**. The real fun starts when you realise that by default they use the *same* terminal−database entry, so the fact that the `kbs` capability is associated to an ASCII DEL makes all correctly behaving applications produce the same behaviour for the **Backspace** and **Delete** keys in gnome−terminal. The simple statement

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

can solve the problem in this case for correctly behaving applications. Well, not always, because your system could lack an entry in the terminal database named `gnome`, in particular if it is not very up−to−date.

In any case, this is not always a solution: if, for instance, you have a Red Hat 7.0 distribution, your gnome−terminal behaves like a console. But beware: if you upgraded your desktop using the Helix distribution, then your gnome−terminal behaves like a pre−7.0 Red Hat.

Just to make easier the following discussion, let us define *standard* a VT100 emulator behaving like the console, and *deviant* one that emits BS for **Backspace** and DEL for **Delete**.[3] Thus, for instance, xterm has always been standard in the Debian distribution, while it switched a couple of times from standard to deviant and viceversa in Red Hat; the behaviour of gnome−terminal is even more erratic. See Section 8 for some information on how to turn a deviant terminal into a standard one.

---

5. What You Should Do When Writing Applications

When you write a console application, be kind to the user and try to understand what comes from the standard input using the following fallback chain:

1. open the right terminfo entry, and try to process the sequence so as to discover whether it has a particular meaning on the current terminal; if so, use the terminfo semantics;
2. use the ASCII intended meaning on line feeds, newlines, tab characters and, of course, BS and DEL. Crossing your finger could also be useful.

---

6. What You Should Do On Your System

Note again that the main issue that confuses people trying to fix their system is that usually they are fixing thing in the wrong place. Since the parts that work often just work by chance, trying to fix the system assuming something is broken will often lead to change correct settings into incorrect settings.

6.1. What Needs to Be Done

6.1.1. Detecting Deviance

The first step towards a clean solution is to know exactly which terminals are deviant and which not. Usually they all behave like the console, and in this case the modifications to get everything working are minimal. If, however, you have some deviant terminal (e.g., a deviant version of gnome−terminal), you will have to treat it in a special way.

The following C one−liner

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

may help you. Put the line into a file named `ascii.c`, compile it with **gcc ascii.c −o ascii**, type **./ascii** and press a key followed by **RETURN**. The program will display the decimal and hexadecimal codes of the ASCII sequence produced (you may want to do a **stty erase ^−** first to get really all the codes). Now you can easily see what **Backspace** key does: if it emits a DEL (127), you have a standard emulator, if it emits a BS (8) you have a deviant one.

6.1.2. Distinguishing Between Emulators

If you have some deviant terminal emulator, you must distinguish it from the standard ones. Theoretically, this should not be a problem because there are different entries in the terminal database for terminals with different sequences (the entry used depends on the value of the `TERM` variable).

Here we take the approach that the `gnome` entry should be used for all deviant VT100 emulators, and the `xterm` entry for the standard ones. This is in line with several distributions (except a few cases like RedHat d5.0, where the `xterm` entry is deviant).

However, gnome−terminal uses by default the same entry as xterm, so if one is deviant and the other one is not you will need to find a way to tell them apart. The option `termname` of gnome−terminal allows the user to set the `TERM` variable to a more sensible name. However, in older versions of gnome−terminal the option does not work. Moreover, sometimes it is not easy to modify the way gnome−terminal is started.

A good idea here is to exploit the fact that gnome−terminal sets the `COLORTERM` variable to `gnome-terminal`. Thus, by adding a simple test to the shell configuration files we can fix the `TERM` variable.

6.1.3. Fixing the Terminal Database

Our problem now is that the terminal database could lack a `gnome` entry for deviant terminals (this happens on a number of termcap and terminfo versions). Recent terminfo databases have an entry `gnome`, but, in any case, since gnome–terminal behaves essentially like xterm modulo our famous two keys, it is possible to automagically generate a brand new correct entry.

6.1.4. Fixing the Shell Behaviour

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.

Moreover, if you want less and other application that do raw line input to work correctly, you must convince the shell that under a deviant terminal emulator the erase character is BS, and not DEL (in the other case the **Backspace** key is already emitting DEL, so we do not have to do anything). This can be done using the command **stty**.

6.2. How to Do It

These fixes have some drawbacks. First, they work only for the specified terminals. Second, in theory (but this is unlikely to happen) they could confuse the readline library on other terminals. Both limitations are however mostly harmless.

First of all, check with **infocmp gnome** whether you already have a `gnome` entry in your terminfo database (we will fix termcap later). If the entry does not exist, the following command

```
bash$ tic <(infocmp xterm |\
        sed 's/xterm|/gnome|/' |\
        sed 's/kbs=\\177,/kbs=^H,/' |\
        sed 's/kdch1=\\E\[3~,/kdch1=\\177,/')
```

will create a correct one in `~/.terminfo`. If the same command is launched by the root, it will generate the entry in the global database (you can override this behaviour by setting `TERMINFO` to `~/.terminfo`). Note that if your `xterm` entry is already deviant (e.g., you have a Red Hat d5.0) the script will copy it unchanged, which is exactly what we want.

Now, add the following snippet to `~/.inputrc`[4]:

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

This line teaches the readline library how to manage your standard **Delete** key for standard emulators, and with a bit of luck it should not interfere with other terminals. However, now we must also explain to the library the meaning of the DEL character on deviant terminals, for instance by adding

```
$if term=gnome
DEL: delete-char
Meta-DEL: kill-word
```

```
"\M-\C-?": kill-word
$endif
```

to ~/.inputrc. If xterm is deviant, too, you must add other three lines for it. On the other hand, if no terminal emulator is deviant this part is not needed. All these changes can be made global by altering the /etc/inputrc file.

Note that the conditional assignments make deviant terminal emulators work *given that the TERM variable is set correctly*. To guarantee this, there are a number of techniques. First of all, since the default value of the TERM variable for gnome−terminal is xterm, if all terminals are not deviant then we do nothing. If, however, a terminal that by default uses the xterm entry is deviant you must find a way to set the TERM variable correctly; assume for instance this is true of gnome−terminal.

The simplest way to obtain this effect is to start gnome−terminal with the argument --termname=gnome, for instance by suitably setting the command line in the launcher on the GNOME panel. If however you have an old version, and this method does not work, you can add the lines

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

to your ~/.bashrc configuration file[5]. The assignment is executed only under gnome−terminal, and sets correctly the TERM variable.

Setting the terminal to gnome could prevent **ls** from using colours, as many versions of **ls** do not know that gnome−terminal is colour capable. To avoid this problem, create a configuration file ~/.dircolors with **dircolors −−print−database >~/.dircolors**, and add a line **TERM=gnome** to the configuration file.

We will now generate on−the−fly a suitable termcap entry for deviant terminal emulators; this can be done as follows, always in ~/.bashrc:

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

Finally, we must explain to the terminal device which character is generated by the erase key. Since usually the erase key is expected to backspace, there is a nice trick taken from the Red Hat /etc/bashrc that works: add this to ~/.bashrc:

```
KBS=$(tput kbs)
if [ ${#KBS} -eq 1 ]; then stty erase $KBS; fi
```

It's a simple idea: we read from the terminal database the capability kbs, and set the erase character to its value if it is a single character (which happens in both standard and deviant terminals).

Certain distributions could have fixes already in place in the system−wide /etc/inputrc configuration file. In this case you

can eliminate redundant lines from your `~/.inputrc`.

6.3. Fixing for tcsh

In the case of the tcsh, the fixes go all in `~/.tcshrc`, and follow the same rationale as the ones for the bash:

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

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

if ($?TERM) then
   if ($TERM == "gnome") then
      setenv TERMCAP \
       "`infocmp -C gnome | grep -v '^#' | tr '\n\t' '  ' | sed 's/\\  //g' | sed s/::/:/g`"
      bindkey "^?" delete-char
      bindkey "^[^?" delete-word
      bindkey "\377" delete-word
   endif
endif

set KBS=`tput kbs`
if (${%KBS} == 1) then
   stty erase $KBS
endif
```

The second part must be replicated for every deviant terminal. Of course, if a termcap entry already exists it is not necessary to generate it.

7. What If Nothing Works

The first thing to do is understanding which ASCII codes are produced by a certain key using the C one−liner.

Once you know which sequences are produced, you must check the current terminfo entry with **infocmp** (don't be scared by the amount of information printed!) and be sure that the `kbs` and `kdch1` capabilities correspond to the right sequences (that is, the one produced by the respective keys). Moreover, you must check with **stty −a** that the erase character is the one emitted by the **Backspace** key (note that `^H` represent BS whereas `^?` represents DEL).

If there is a mismatch, there can be several different reason: wrong content of the `TERM` variable, wrong entry of the terminal database, wrong terminal emulation under X. I hope at this point you have enough information to dig the solution autonomously.

If different applications behave in different ways, it is likely that some of them are using the terminal database correctly, and some are not. Remember that the fact that the keys produce the right behaviour in a certain application does not mean that the application is using correctly the terminal database they could work just by chance. If you want to have an independent check, you can try whether the **ne** editor works. **ne** uses all terminal capabilities, including `kbs` and `kdch1`, and uses intended meaning only as a last resource.

---

8. More Hacking

So, you're not happy with the information you got. In this case, there is even more hacking you can do on the Backspace/Delete issue, using suitable commands that get or set the way X and the console handle keys.

It could happen that, for some reason, what I said talking about X is not true, that is, X does *not* translate keycode 22 to keysym BackSpace and keycode 107 to keysym Delete (or even that, on your particular keyboard, the keycodes associated to **Backspace/Delete** are not 22 and 107). To be sure of that, you need to use **xev**, a simple X application that will display the keycode and keysym associated to the key you press. If anything goes wrong, there are several ways you can fix the problem: the easy, temporary way is to use **xmodmap**, a command that lets you change many settings related to X keyboard handling. For instance,

```
xmodmap -e "keycode 22 = BackSpace"
xmodmap -e "keycode 107 = Delete"
```

will set correctly the keysyms (assuming that 22 and 107 are the correct keycodes for you). In case you want to do some changes permanently, you can play with the resources `vt100.backArrowKey`, `vt100.translations` and `ttyModes` of xterm (and similar terminal applications) in the configuration file `~/.Xdefaults`. One possibility, for instance, is

```
XTerm.VT100.Translations: \
        <Key>BackSpace: string(0x7F)\n\
        <Key>Delete:    string("\033[3~")
```

You should take a look at the xterm man page for more information.

The program that does for the console what **xev** does for X is **showkeys**: it will dump the console keycodes of the keys you press. Combining **showkeys** with **dumpkeys**, which will print on standard output the console keymap, you can easily fix mismatches between keycodes and keysyms. Analogously to **xmodmap**, **loadkeys** can then fix single associations, or load entirely new console keymaps. With it, you can even change the string associated to a given keysym. If you want to record these changes, you will have to define a new keymap for the console (you should have a look at the system keymaps, usually located in `/lib/kbd`).

---

9. Conclusions

The fixes suggested here should solve to a large extent the problem of deleting text you wrote (however, they do not help in creating other text `:)`).

There is a small bug in the whole setting: if you're using the `COLORTERM` trick and you start xterm from gnome−terminal, the former will get `TERM` set to `gnome`. This inconvenience is, of course, mostly harmless, and does not occur if you simply started gnome−terminal with `TERM` suitably set.

Another nontrivial problem that essentially has no solution is the one concerning remote connections: if you connect to a host whose terminal database is incoherent with yours, you will have to set up things manually.

Finally, it should be noted that the fixes will not work for broken applications (for instance, applications ignoring the `kbs` key capability). There is little to do in this case, as fixing for one broken application will likely break all well−behaving ones.

Notes

[1]
This claim has been asserted/disputed several times commenting this document. If you have any definitive information on this subject, please write me.

[2]
Some programs rely on the terminal driver for input line editing, such as deleting characters or words. With **stty**, you can tell the terminal driver what character it should use to delete the character to the left of the cursor (the *erase* character). You can check your current settings with **stty −a** and set them with **stty erase *character***.

[3]
Also these definitions have been asserted/disputed several times commenting this document. If you have any definitive information on this subject, please write me.

[4]
On older version of the bash, you must remember to set `INPUTRC` suitably, for instance adding

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

to your `~/.profile` (or whichever file is read just by login shells).

[5]
More precisely, to the shell configuration file that is read in every shell, not only in login shells. The right file depend on startup sequence of your bash.