# Petit guide pratique des touches Backspace et Suppr sous Linux

Version française du petit guide *Linux Backspace/Delete mini-HOWTO*

Sebastiano Vigna <vigna@acm.org>

Adaptation française: Baptiste Mélès

Préparation de la publication de la v.f.: Jean-Philippe Guérard

Version : 1.6.fr.1.0

5 novembre 2004

Historique des versions

| | | |
|---|---|---|
| Version 1.6.fr.1.0 | 2004-11-05 | BM, XX, JPG |
| Première adaptation française. | | |
| Version 1.6 | 2002-01-19 | SV |
| J'ai inclus de nombreux commentaires d'Alex Boldt et Chung-Rui Kao | | |
| Version 1.5 | 2001-05-03 | SV |
| Mise à jour pour les nouvelles distributions et l'astuce du tput. *Updated for new distros and the tput trick.* | | |
| Version 1.4 | 2000-12-07 | SV |
| Mise à jour pour les conflits de Red Hat 7.0 et de Helix Gnome. *Updated for Red Hat 7.0 and Helix Gnome conflicts.* | | |
| Version 1.3 | 2000-10-15 | SV |
| Changement du nom. *Name change.* | | |
| Version 1.2 | 2000-10-15 | SV |
| Mise à jour. Ajout de la section « Que faire si rien ne marche ». *Updated. Added "What If Nothing Works" section.* | | |
| Version 1.1 | 2000-09-13 | SV |
| Ajout de réglages pour tcsh. *Added tcsh fixes* | | |
| Version 1.0 | 2000-09-05 | SV |
| Première version. *First release* | | |

## Table des matières

1. Introduction .......... 2
2. Comment les touches sont converties en actions .......... 2
3. Pourquoi ça ne marche pas (toujours) .......... 3
4. X .......... 4
5. Ce que vous devriez faire quand vous écrivez des applications .......... 5
6. Ce que vous devriez faire sur votre système .......... 5
   6.1. Ce qui doit être fait .......... 5
   6.2. Comment le faire .......... 6
   6.3. Réglage de tcsh .......... 8
7. Que faire si rien ne marche .......... 9
8. Plus de bidouillage .......... 9
9. Conclusions .......... 10

# 1. Introduction

Tout utilisateur de Linux a tôt ou tard été confronté à une situation dans laquelle il semblait impossible d'avoir des touches **Backspace** et **Suppr** qui fonctionnent, en console comme sous X. Cette page explique ce phénomène et y suggère des solutions. Dans l'ensemble, les notions abordées ici sont indépendantes de la distribution utilisée : mais en raison de l'étendue des différences entre les fichiers de configuration du système selon les distributions, j'essaierai de donner au lecteur suffisamment de connaissances pour concevoir ses propres réglages en cas de besoin.

Je suppose que la touche **Backspace** doit revenir d'un caractère puis effacer le caractère qui est placé sous le curseur. Au contraire, la touche **Suppr** doit effacer le caractère qui est placé sous le curseur, sans que ce dernier ne bouge. Si vous pensez que la fonction des deux touches doit être intervertie, en dépit du fait que la majorité des claviers fournisse une flèche orientée vers la *gauche* ($\leftarrow$) sur la touche **Backspace**, alors cette page ne vous donnera pas de solutions immédiates, mais vous pourrez néanmoins trouver très utiles les explications données ici.

Je suppose également que les réglages ne devraient modifier que des fichiers locaux (d'un utilisateur). Aucune partie standard de la distribution ne devrait être altérée. Enfin, ce document traite de la façon dont il faut configurer votre système pour que les applications reçoivent les bons événements clavier. Si une application décide d'interpréter de tels événements d'une façon idiosyncratique, le seul réglage possible est de reconfigurer l'application.

### Note

Depuis la première version de ce petit guide pratique, les choses se sont encore embrouillées. Différentes distributions d'un même émulateur de terminal (par exemple gnome-terminal tel qu'il est fourni par Red Hat 7.0, Helix Code/Ximian ou même Red Hat#7.1) génèrent des séquences ASCII. En raison de cette dissonnance, les actuelles bases de données de terminal correspondent encore moins aux émulateurs de terminal qu'elles sont supposées décrire. Afin de donner un fondement solide aux explications qui suivent, nous supposons foncièrement corrects les paramètres proposés par la politique relative au clavier adoptée par Debian [http://www.debian.org/doc/debian-policy/].

# 2. Comment les touches sont converties en actions

Lorsque l'on appuie sur une touche, un nombre de composants matériels et logiciels coopèrent afin de garantir que la signification de la touche que l'on a dans l'esprit (par exemple, d'émettre un certain caractère) concorde avec le comportement effectif de la touche. Je me concentrerai sur l'aspect logiciel (dans la mesure où notre contrôle sur le matériel est inexistant), et en particulier, provisoirement, sur les événements clavier liés à la sortie en console.

1. L'appui sur une touche génère de purs *codes de balayage* (*scancodes*) du clavier ; ces codes de balayage sont ensuite transformés en un *code touche* (*keycode*). Sur un système i386, en général, la touche **Backspace** émet 14 et la touche **Suppr** émet 111.

2. Les codes touche sont traduits par la bibliothèque clavier en un *symbole clavier* (*keyboard symbol : keysym*) eu moyen de la définition du clavier chargée par l'utilisateur. Si vous regardez dans votre base de données de clavier (par exemple dans `/lib/kbd/`), vous découvrirez plusieurs définitions pour plusieurs ordinateurs, différentes répartitions et probablement différentes interprétations des mêmes touches (par exemple, on peut vouloir que les deux touches **Alt** se comportent vraiment comme deux modificateurs distincts). La répartition du clavier de la console Linux assigne la touche symbolique Sup-

pr au code touche 14 et la touche symbolique Removeau code touche 111. Cela peut sembler étrange, mais la console Linux émule un terminal VT100, et c'est ainsi que vont les choses ici-bas. [1]

3. Notre voyage touche déjà à sa fin. Les applications en console lisent des séquences ASCII, pas des touches symboliques. La console doit donc lire des touches symboliques et les traduire en des séquences ASCII chiffrant convenablement les touches. Bien sûr, cette opération doit être effectuée d'une manière qui soit compréhensible par les applications. En l'occurrence, dans la console Linux, la touche symbolique Suppr est mappée vers le code ASCII 127 (DEL), la touche symbolique Remove vers une séquence d'échappement adéquate, et la touche symbolique BackSpace vers le code ASCII 8 (BS).

4. Enfin, nous devons en quelque sorte revenir sur nos pas et traduire les séquences ASCII générées par chaque touche en capabilité de touche. Cet objectif est atteint par une *base de données de terminal*, qui contient, pour chaque type de terminal, le mappage inversé des séquences de caractères vers les capabilités de touche (qui sont essentiellemnt un sous-ensemble des touches symboliques). [2]

### Note

Malheureusement, il y a deux bases de données de terminal « standard », termcap et terminfo. Selon votre distribution, vous pouvez tout aussi bien utiliser l'une de ces deux là, ou la base de donnés peut encore dépendre de l'application. Nos explications se concentreront sur la base de données terminfo, qui est plus récente, mais les réglages suggérés ici prennent les deux en considération.

En l'occurrence, dans la console Linux, **F1** génère un échappement suivi de `[[A`, qui peut être traduit par la capabilité `key_f1` en regardant dans l'entrée de la base de données du terminal de la console (essayez **infocmp linux** si vous voulez regarder l'entrée). On peut trouver une très bonne explication des bases de données du terminal dans le manuel du termcap de GNU. En général, les applications Linux utilisent la base de données terminfo, plus récente, contenue dans le paquetage ncurses.

D'une façon qui n'est peut-être pas si étonnante, l'entrée de terminfo en console Linux mappe DEL vers la capabilité `kbs` (touche backspace), et Échap suivi de `[3~` vers la capabilité `kdch1` (touche « delete-one-char »). Même si vous pouvez trouver étrange que la touche **Backspace** émette un DEL, la base de données de terminal remet tout à sa place, et les applications qui se comportent correctement interpréteront DEL comme la capabilité `kbs`, effaçant ainsi le caractère situé à gauche du curseur.

# 3. Pourquoi ça ne marche pas (toujours)

J'espère qu'au point où nous en sommes, le problème de base est clair : il y a un goulet d'étranglement entre le clavier et les applications console, à savoir, le fait qu'ils ne puissent communiquer que par séquences ASCII. Les touches spéciales sont ainsi d'abord traduites de touches symboliques en séquences, puis de séquences en capabilités de touche. Comme des consoles différentes ont des idées divergentes sur l'aspect que doit prendre cette traduction, nous avons besoin d'une base de données de terminal. Le système marcherait parfaitement, à un petit problème près : la base de données n'est pas toujours configurée correctement, et tout le monde ne l'utilise pas.

Les applications peuvent avoir un moyen de savoir quelle entrée de base de données utiliser : ce qui le rend réalisable, c'est une configuration adéquate de la variable d'environnement `TERM`. Dans certains cas, il y a une dissonnance entre l'émulateur de terminal et le contenu de l'entrée de la base de données suggérée par `TERM`.

[1] Cette affirmation a été confirmée et discutée plusieurs fois dans les commentaires sur ce document. Si vous avez quelque information décisive à ce sujet, prière de m'écrire.

[2] Certains programmes s'en remettent au pilote du terminal pour l'édition de la ligne d'entrée, comme l'effacement de caractères ou de mots. Avec **stty**, vous pouvez dire au pilote du terminal quel caractère il doit utiliser pour effacer le caractère placé à gauche du curseur (le caractère *erase*). Vous pouvez contrôler vos paramètres courants avec **stty a** et les choisir avec **stty erase** ***`character`***.

En outre, de nombreuses applications *n'utilisent pas* la base de données de terminal (ou au moins pas entièrement), et considèrent les codes ASCII BS et DEL avec un sens qu'elles ne font que présumer : par conséquent, sans regarder dans la base de données, elles leur assignent une signification (en général, bien sûr, la signification est d'effacer le caractère situé avant ou sous le curseur). Ainsi notre beau schéma est complètement détruit (tout utilisateur de Linux en fait l'amère expérience). En l'occurrence, bash suppose que DEL doit faire un backward-delete-char, c'est-à-dire backspace.

Partant, tout fraîchement installé, il permet que la touche **Backspace** fonctionne en console comme on peut s'y attendre, mais uniquement à cause de la succession de deux renversements ! Bien sûr, la touche **Suppr** ne fonctionne pas. La cause de ce phénomène est que bash ne cherche pas dans la base de données de terminal la capabilité `kdch1`.

Juste pour illustrer la façon dont les choses se sont embrouillées, considérez le script **fix_bs_and_del** fourni avec la distribution Red Hat (et peut-être d'autres). Il assigne à la volée la touche symbolique BackSpace à la touche **Backspace**, et la touche symbolique Suppr à la touche **Suppr**. Désormais le shell fonctionne ! Malheureusement, tous les programmes qui se reposaient sur l'association correcte de la génération de touches symboliques et de mappages de la base de données de terminal ne marchent plus du tout maintenant, car la touche symbolique Suppr est mappée par la base de données terminfo vers DEL, et celui-ci vers la capabilité de touche `kbs`, de telle sorte que dans de tels programmes, les deux touches produisent un backspace.

# 4. X

Sous X, la situation n'est pas vraiment différente. Il y a juste une couche différente, à savoir que le système X Window traduit les scancodes en ses propres touches symboliques, qui sont bien plus variées et plus précises que celles de la console, et les fournit aux applications (au fait, c'est la raison pour laquelle XEmacs n'est pas tourmenté par ce problème : X traduit le code touche 22 par la touche symbolique BackSpace et le code touche 107 par la touche symbolique Suppr, puis l'utilisateur peut facilement assigner à ces touches symboliques le comportement désiré). Bien sûr, un programme d'émulation de terminal (en général un émulateur de VT100 dans le monde X) doit traduire les touches symboliques X en séquences ASCII, de telle sorte que nous en revenons à notre douloureuse affaire.

Plus précisément, en général xterm se comporte exactement comme la console (i.e. il émet les mêmes séquences ASCII), mais, par exemple, gnome-terminal dans la Red Hat <7.0 or $_{\geq 7.1}$ émet BS pour **Backspace** et DEL pour **Suppr**. Là où ça commence à devenir drôle, c'est quand vous vous rendez compte qu'ils utilisent par défaut la *même* entrée de base de donnée de terminal ; ainsi, le fait que la capabilité `kbs` soit associée à un DEL ASCII fait que toutes les applications qui se comportent correctement produisent le même comportement pour les touches **Backspace** et **Suppr** dans gnome-terminal. La simple commande

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

peut résoudre le problème, dans ce cas, pour les applications qui se comportent correctement. Bon, pas toujours, parce qu'il pourrait manquer une entrée appelée `gnome` à votre système dans la base de données de terminal, en particulier s'il n'est pas parfaitement à jour.

Dans certains cas, ce n'est pas toujours une solution : si, par exemple, vous avez une distribution Red Hat 7.0, votre gnome-terminal se comporte comme une console. Mais attention : si vous avez mis à jour votre bureau en utilisant la distribution Helix, alors, votre gnome-terminal se comporte comme une Red Hat antérieure à la version 7.0.

À la seule fin de faciliter la compréhension de ce qui va suivre, appelons *standard* un émulateur de VT100 se comportant comme la console, et *non-standard* un qui émet BS pour **Backspace** et DEL pour **Suppr**[3].

[3]Même ces définitions ont été confirmées et discutées de nombreuses fois au sujet de ce document. Si vous avez une information décisive sur ce sujet, vous êtes prié de m'écrire.

Ainsi, par exemple, xterm a toujours été standard dans la distribution Debian, tandis qu'il a oscillé à de nombreuses reprises entre les états standard et non-standard dans la Red Hat ; le comportement de gnome-terminal est encore plus erratique. Voir Section 8, « Plus de bidouillage » pour quelques renseignements sur la façon de rendre standard un terminal non-standard.

# 5. Ce que vous devriez faire quand vous écrivez des applications

Lorsque vous écrivez une application en console, soyez gentils pour l'utilisateur et essayez de comprendre ce qui vient de l'entrée standard au moyen de la procédure de restauration suivante :

1. ouvrez la bonne entrée terminfo et essayez de traiter la séquence afin de découvrir si elle a une signification particulière dans le terminal courant ; si c'est le cas, utilisez la sémantique terminfo ;

2. utilisez la signification ASCII voulue sur les contenus de ligne, les retours chariot, les tabulations et, bien sûr, BS et DEL. Croiser les doigts pourrait également se révéler utile.

# 6. Ce que vous devriez faire sur votre système

Remarquez encore que le problème qui égare souvent les gens qui essayent de régler leur système est qu'il règlent les choses au mauvais endroit. Puisque les parties qui fonctionnent ne fonctionnent souvent que par chance, essayer de régler le système en supposant que quelque chose est cassé mènera souvent à changer des paramètres corrects en paramètres incorrects.

## 6.1. Ce qui doit être fait

### 6.1.1. La détection des comportements non-standards

La première étape d'une solution propre est de savoir exactement quels terminaux sont standards et lesquels ne le sont pas. En général, ils se comportent tous comme la console, et dans ce cas les modifications à faire pour que tout fonctionne bien sont minimes. Si, cependant, vous avez un terminal non-standard (par exemple, une version non-standard de gnome-terminal), vous aurez à le traiter d'une façon spécifique.

Le programme suivant, composé d'une ligne de C,

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

pourrait vous aider. Mettez cette ligne dans un fichier intitulé `ascii.c`, compilez-le avec **gcc ascii.c -o ascii**, tapez **./ascii** et appuyez sur une touche suivie de **Entrée**. Le programme affichera les codes décimal et hexadécimal de la séquence ASCII produite (vous pourriez d'abord vouloir faire un **stty erase ^-** pour obtenir vraiment tous les codes). Vous pouvez désormais voir facilement ce que la touche **Backspace** fait : si elle émet un DEL (127), vous avez un émulateur standard, si elle émet un BS (8), vous en avez un non-standard.

### 6.1.2. Distinguer les émulateurs

Si vous avez un émulateur de terminal non-standard, vous devez le distinguer des émulateurs standards. Théoriquement, cela ne devrait pas être un problème parce qu'il y a différentes entrées dans la base de données de terminal avec différentes séquences (l'entrée utilisée dépend de la valeur de la variable `TERM`).

Nous en venons ici au fait que l'entrée `gnome` doit être utilisée pour tous les émulateurs de VT100 non-standards, et l'entrée`xterm` pour les émulateurs standards. C'est avec de nombreuses distributions (sauf quelques cas comme la Red Hat $\leq$5.0, où l'entrée `xterm` est non-standard).

Cependant, gnome-terminal utilise par défaut la même entrée que xterm ; ainsi, si l'un est non-standard et que l'autre est standard, vous devrez trouver un moyen de le leur dire séparément. L'option `termname` de gnome-terminal permet à l'utilisateur de choisir la variable `TERM`. Cependant, dans d'anciennes versions de gnome-terminal, l'option ne fonctionne pas. En outre, il n'est parfois pas facile de modifier la façon dont gnome-terminal est démarré.

C'est une bonne idée que d'exploiter ici le fait que gnome-terminal règle la variable `COLORTERM` sur `gnome-terminal`. Ainsi, par le simple ajout d'un test aux fichiers de configuration du shell, nous pouvons régler la variable `TERM`.

### 6.1.3. Réglage des bases de données de terminal

Notre problème est désormais qu'il pourrait manquer à la base de données de terminal une entrée `gnome` pour les terminaux non standards (c'est le cas dans un certain nombre de versions de termcap et de terminfo). Les bases de données terminfo récentes ont une entrée `gnome`, mais, dans certains cas, comme gnome-terminal se comporte dans l'ensemble comme xterm, à nos deux fameuses touches près, il est possible de générer automatiquement une entrée flambant neuve et correcte.

### 6.1.4. Réglage du comportement du shell

La bibliothèque readline, utilisée par bash et de nombreux autres programmes pour lire la ligne d'entrée, peut être personnalisée de façon à reconnaître les séquances spécifiques de caractères. La personnalisation peut également dépendre de la variable `TERM`, si bien qu'une fois que l'on peut distinguer les terminaux, on peut régler finement le clavier.

Par ailleurs, si vous voulez que less et autres applications en mode texte marchent correctement, vous devez convaincre le shell du fait que sous un émulateur de terminal non-standard, le caractère d'effacement soit BS, et pas DEL (dans le cas contraire, la touche **Backspace** émet déjà DEL, donc nous n'avons rien à faire). On peut le faire à l'aide de la commande **stty**.

## 6.2. Comment le faire

#### Attention

Ces réglages peuvent avoir quelques effets pervers. D'abord, ils ne marchent que pour les terminaux spécifiés. Ensuite, en théorie (mais il est peu probable que cela arrive) ils pourraient embrouiller la bibliothèque readline sur d'autres terminaux. Mais ces deux limitations sont en général sans inoffensives.

Tout d'abord, vérifiez avec **infocmp gnome** si vous avez déjà une entrée `gnome` dans votre base de données terminfo (nous réglerons plus tard le cas de termcap). Si l'entrée n'existe pas, la commande suivante

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

en créera une correcte dans `~/.terminfo`. Si la même commande est lancée par l'administrateur, elle générera l'entrée dans la base de données globale (vous pouvez transgresser ce comportement en réglant

TERMINFO sur ~/.terminfo). Notez que si votre entrée xterm est déjà non-standard (par exemple, vous avez une Red Hat ≤5.0), le script la copiera sans la changer, et c'est précisément ce que nous voulons.

Ajoutez maintenant le fragment suivant à ~/.inputrc[4] :

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

Cette commande apprend à la bibliothèque readline comment gérer votre touche standard **Suppr** pour des émulateurs standards, et avec un peu de chance, cela ne devrait pas interférer avec d'autres terminaux. Cependant, nous devons maintenant expliquer à la bibliothèque la signification du caractère DEL dans les terminaux non-standards, en ajoutant par exemple

```
$if term=gnome
DEL: delete-char
Meta-DEL: kill-word
"\M-\C-?": kill-word
$endif
```

à ~/.inputrc. Si xterm est non-standard également, vous devez ajouter trois autres lignes pour lui. Inversement, si tous les terminaux sont standards, cette partie n'est pas requise. Toutes ces modifications peuvent être étendues en changeant le fichier /etc/inputrc.

Remarquez que les assignements conditionnels font que les terminaux non-standards fonctionnent *pour autant que la variable TERM soit configurée correctement*. Pour le garantir, il y a plusieurs techniques. D'abord, comme la valeur par défaut de la variable TERM est, pour gnome-terminal, xterm, si tous les terminaux sont standards, nous n'avons rien à faire. Si, cependant, un terminal qui utilise par défaut l'entrée xterm est non-standard, vous devez trouver un moyen de régler correctement la variable TERM ; supposez par exemple que ceci est vrai de gnome-terminal.

Le moyen le plus simple d'obtenir cet effet est de démarrer --termname=gnome, par exemple en réglant correctement la ligne de commande dans le lanceur de la barre d'outils GNOME. Si vous avez cependant une version plus ancienne et que cette méthode ne fonctionne pas, vous pouvez ajouter les lignes

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

à votre fichier de configuration ~/.bashrc[5]. L'assignement n'est exécuté que sous gnome-terminal, et règle correctement la variable TERM.

### Note

Régler le terminal sur gnome pourrait empêcher **ls** d'utiliser des couleurs, car de nombreuses versions de **ls** ignorent que gnome-terminal supporte les couleurs. Afin d'éviter ce problème, créez

[4] Sur les anciennes versions de bash, vous devez vous souvenir de régler correctement INPUTRC, par exemple en ajoutant

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

à votre ~/.profile (ou au fichier, quel qu'il soit, qui n'est lu que par les shells de login).

[5] Plus précisément, au fichier de configuration du shell qui est lu dans n'importe quel shell, pas seulement dans les shells de login. Le bon fichier dépend de la séquence de démarrage de votre bash.

un fichier de configuration `~/.dircolors` avec **dircolors --print-database >~/.dircolors**, et ajoutez une ligne **`TERM=gnome`** au fichier de configuration.

Nous allons maintenant générer à la volée une entrée termcap qui convienne pour les émulateurs de terminaux non-standards ; on peut le faire comme suit, toujours dans `~/.bashrc` :

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

Enfin, nous devons expliquer au périphérique de terminal quel caractère est généré par la touche d'effacement. Comme en général la touche d'effacement est supposée faire un retour arrière, il y a une astuce sympathique et efficace tirée du `/etc/bashrc` de Red Hat : ajoutez ceci à `~/.bashrc` :

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

C'est une idée toute simple : nous lisons la capabilité `kbs` à partir de la base de données de terminal, et réglons le caractère d'effacement sur sa valeur si c'est un caractère simple (ce qui est le cas aussi bien dans les terminaux standards que dans les non-standards).

### Note

Certaines distributions peuvent avoir des réglages déjà en place dans le fichier de configuration `/etc/inputrc`, valable à l'échelle du système tout entier. Dans ce cas, vous pouvez éliminer les lignes redondantes de votre `~/.inputrc`.

## 6.3. Réglage de tcsh

Dans le cas de tcsh, les réglages vont tous dans `~/.tcshrc`, et suivent la même logique que pour 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
```

La seconde partie doit être répétée pour chaque terminal non-standard. Bien sûr, si une entrée termcap existe déjà, il n'est pas nécessaire de la générer.

# 7. Que faire si rien ne marche

La première chose à faire est de comprendre quels codes ASCII sont produits par une certaine touche utilisant le programme d'une ligne de C.

Une fois que vous savez quelles séquences sont produites, vous devez vérifier l'entrée terminfo en cours avec **infocmp** (ne soyez pas effrayé par la somme des informations affichées !) et assurez-vous que les capabilités `kbs` et `kdch1` correspondent aux bonnes séquences (c'est-à-dire celles produites par les touches respectives). En outre, vous devez vérifier avec **stty -a** que le caractère d'effacement est celui qui est émis par la touche **Backspace** (remarquez que `^H` représente BS tandis que `^?` représente DEL).

S'il y a dissonnance, il peut y avoir plusieurs raisons : mauvais contenu de la variable `TERM`, mauvaise entrée de la base de données de terminal, mauvaise émulation de terminal sous X. J'espère qu'arrivé à ce point, vous avez suffisamment d'informations pour creuser la solution par vous-même.

### Note

Si différentes applications se comportent différemment, il est possible que certaines d'entre elles utilisent correctement la base de données de terminal, et d'autres non. Rappelez-vous que le fait que les touches adoptent le bon comportement dans une application donnée ne signifie pas que l'application utilise correctement la base de données de terminal— elles peuvent ne fonctionner que par chance. Si vous voulez avoir une vérification indépendante, vous pouvez essayer de voir si l'éditeur **ne** [http://ne.dsi.unimi.it/] fonctionne ou non. **ne** utilise toutes les capabilités du terminal, y compris `kbs` et `kdch1`, et n'utilise la signification désirée qu'en dernier recours.

# 8. Plus de bidouillage

Ainsi, vous n'êtes pas satisfait par toutes ces informations. Dans ce cas, vous pouvez faire encore plus de bidouillage sur la sortie de Backspace et de Suppr, en ayant recours aux commandes qui conviennent pour observer ou régler la façon dont X et la console traitent les touches.

Il peut arriver que, pour quelque raison, ce que j'ai dit sur X ne soit pas vrai, c'est-à-dire que X ne traduise *pas* le code touche 22 en touche symbolique BackSpace, ni le code touche 107 en touche symbolique Suppr (ou même que, sur votre clavier particulier, les codes touche associés à **Backspace** et **Suppr** ne soient pas 22 et 107). Afin de vous en assurer, vous devez utiliser **xev**, une application X simple qui affichera le code touche et la touche symbolique associés à la touche que vous enfoncez. Si quelque chose va de travers, il y a plusieurs façons de régler le problème : le moyen simple et provisoire est d'utiliser **xmodmap**, une commande qui vous permet de changer de nombreux réglages relatifs au traitement du clavier par X. Par exemple,

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

réglera correctement les touches symboliques (en supposant que 22 et 107 sont les codes touches corrects pour vous). Dans le cas où vous voudriez opérer quelques modifications sur le long terme, vous pouvez jouer avec les ressources `vt100.backArrowKey`, `vt100.translations` et `ttyModes` de xterm (et les applications de terminal similaires) dans le fichier de configuration `~/.Xdefaults`. Une possibilité, par exemple, consiste en ceci :

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

Vous devriez jeter un coup d'œil dans la page de manuel de xterm pour obtenir plus d'informations.

Le programme faisant pour la console ce que **xev** fait pour X est **showkeys** : il renverra les codes touche de console des touches que vous enfoncez. En combinant **showkeys** avec **dumpkeys**, qui affichera sur la sortie standard le mappage clavier de la console, vous pouvez facilement régler les dissonnances entre les codes touche et les touches symboliques. Parallèlement à **xmodmap**, **loadkeys** peut ensuite régler des associations isolées, ou charger des mappages clavier entièrement nouveaux. Avec cela, vous pouvez même changer la chaîne associée à une touche symbolique donnée. Si vous voulez enregistrer ces modifications, vous devrez définir un nouveau mappage clavier pour la console (vous devriez jeter un coup d'œil sur les mappages clavier du système, en général situés dans `/lib/kbd`).

# 9. Conclusions

Les réglages suggérés ici devraient résoudre une grande étendue de problèmes liés à l'effacement du texte que vous avez écrit (cependant, ils n'aident pas à en créer du nouveau **:)** ).

Il y a un petit bug dans le réglage tout entier : si vous utilisez l'astuce du `COLORTERM` et que vous démarrez xterm depuis gnome-terminal, le premier verra `TERM` se régler sur `gnome`. Cet inconvénient est, bien sûr, en général complètement inoffensif, et n'a pas lieu si vous avez tout simplement démarré gnome-terminal avec un `TERM` convenablement réglé.

Un autre problème non trivial, et qui n'a pas vraiment de solution, est celui qui concerne la connexion à distance : si vous vous connectez sur un hôte dont la base de données de terminal est incohérente avec la vôtre, vous aurez à configurer les choses à la main.

Enfin, il faut observer que les réglages ne marcheront pas pour des applications cassées (par exemple, des applications ignorant la capabilité de touche `kbs`). Il n'y a pas grand chose à faire dans ce cas, car régler le problème d'une application cassée présente de fortes chances de casser toutes celles qui se comportent correctement.