The Linux keyboard and console HOWTO

Table of Contents

Table of Contents

The Linux keyboard and console HOWTO

Andries Brouwer, `aeb@cwi.nl`

v2.8, 25 February 1998

This note contains some information about the Linux keyboard and console, and the use of non−ASCII characters. It describes Linux 2.0.

6.The console character sets

7.Console switching

8.Ctrl−Alt−Del and other special key combinations

9.How to get out of raw mode

10.The keyboard LEDs

11.The TERM variable

12.How to make other programs work with non−ASCII chars

13.What precisely does XFree86−2.1 do when it initializes its keymap?

1.Useful programs

The following packages contain keyboard or console related programs.

`kbd-0.95.tar.gz` contains `loadkeys`, `dumpkeys`, `showkey`, `setmetamode`, `setleds`, `setfont`, `showfont`, `mapscrn`, `kbd_mode`, `loadunimap`, `chvt`, `resizecons`, `deallocvt`, `getkeycodes`, `setkeycodes`.

`util-linux-2.6` contains `setterm`, `kbdrate`. (Yes, the `more` in `util-linux-2.6` dumps core due to a name conflict. Preserve your old copy, or use `util-linux-2.5`, or change `savetty' to `my_savetty' in `more.c`.)

`sh-utils-1.12` contains `stty`.

`open-1.4.tgz` contains `open` (that should be renamed to `openvt`). (See also `dynamic-vc-1.1.tar.gz`.)

`SVGATextMode-1.8.tar.gz` contains `SVGATextMode`, a program that obsoletes `resizecons`.

The X distribution contains `xmodmap`, `xset`, `kbd_mode`. (See also X386keybd(1) for the situation under XFree86 1.3, and Xserver(1) for the XKEYBOARD extension under X11R6.)

`termcap-2.0.8.tar.gz` contains `termcap`, an old terminal capabilities data base. `ncurses-1.9.9e.tar.gz` contains the `termlib` data base which obsoletes `termcap`. (However, there are still many programs using `termcap`.)

See loadkeys(1), setleds(1) and setmetamode(1) for the codes generated by the various keys and the setting of leds when not under X. Under X, see xmodmap(1) and xset(1).

See setfont(8) for loading console fonts. Many people will want to load a font like `iso01.f16` because the default font is the hardware font of the video card, and often is a `Code Page 437' font missing accented characters and other Latin−1 symbols.

See setterm(1) and kbdrate(8) for properties such as foreground and background colors, screen blanking and character repeat rate when not under X. Under X, see xset(1), also for key click and bell volume.

The file `/etc/termcap` defines the escape sequences used by many programs addressing the console (or any other terminal). See termcap(5). A more modern version is found in `/usr/lib/terminfo`. See terminfo(5). Terminfo files are compiled by the terminfo compiler `/usr/lib/terminfo/tic`, see tic(1). Their contents can be examined using the program `infocmp`, see infocmp(1). The Linux console sequences are documented in console_codes(4).

2.Keyboard generalities

You press a key, and the keyboard controller sends scancodes to the kernel keyboard driver. Some keyboards can be programmed, but usually the scancodes corresponding to your keys are fixed. The kernel keyboard driver just transmits whatever it receives to the application program when it is in *scancode mode*, like when `X` is running. Otherwise, it parses the stream of scancodes into keycodes, corresponding to key press or key release events. (A single key press can generate up to 6 scancodes.) These keycodes are transmitted to the application program when it is in *keycode mode* (as used, for example, by `showkey`). Otherwise, these keycodes are looked up in the keymap, and the character or string found there is transmitted to the application, or the action described there is performed. (For example, if one presses and releases the `a` key, then the keyboard produces scancodes 0x1e and 0x9e, this is converted to keycodes 30 and 158, and then transmitted as 0141, the ASCII or latin−1 code for `a'; if one presses and releases `Delete`, then the keyboard produces scancodes 0xe0 0x53 0xe0 0xd3, these are converted to keycodes 111 and 239, and then transmitted as the 4−symbol sequence ESC [3 ~, all assuming a US keyboard and a default keymap. An example of a key combination to which an action is assigned is Ctrl−Alt−Del.)

The translation between unusual scancodes and keycodes can be set using the utility `setkeycodes` − only very few people will need it. The translation between keycodes and characters or strings or actions, that is, the keymap, is set using the utilities `loadkeys` and `setmetamode`. For details, see getkeycodes(8), setkeycodes(8), dumpkeys(1), loadkeys(1), setmetamode(1). The format of the files output by `dumpkeys` and read by `loadkeys` is described in keytables(5).

Where it says `transmitted to the application' in the above description, this really means `transmitted to the terminal driver'. That is, further processing is just like that of text that comes in over a serial line. The details of this processing are set by the program `stty`.

3.Console generalities

Conversely, when you output something to the console, it first undergoes the standard tty processing, and then is fed to the console driver. The console driver emulates a VT100, and parses the input in order to recognize VT100 escape sequences (for cursor movement, clear screen, etc.). The characters that are not part of an escape sequence are first converted into Unicode, using one of four mapping tables if the console was not in UTF−8 mode to start with, then looked up in the table describing the correspondence between Unicode values and font positions, and the obtained 8− or 9−bit font indices are then written to video memory, where they cause the display of character shapes found in the video card's character ROM. One can load one's own fonts into character ROM using `setfont`, load the corresponding Unicode map with `loadunimap`, and load a user mapping table using `mapscrn`. More details will be given below.

There are many consoles (called *Virtual Consoles* or *Virtual Terminals*, abbreviated VCs or VTs) that share the same screen. You can use them as independent devices, either to run indendent login sessions, or just to send some output to, perhaps from `top`, or the tail of the system log or so. See below (`Console switching') on how to set them up and switch between them.

4.Resetting your terminal

There is garbage on the screen, or all your keystrokes are echoed as line drawing characters. What to do?

Many programs will redraw the screen when ^L is typed. This might help when there is some modem noise or broadcast message on your screen. The command `clear` will clear the screen.

The command `reset` will reset the console driver. This helps when the screen is full of funny graphic characters, and also if it is reduced to the bottom line. If you don't have this command, or if it does something else, make your own by putting the following two lines in an executable file `reset` in your PATH:

```
#!/bin/sh
echo -e \\033c
```

that is, you want to send the two characters ESC c to the console.

Why is it that the display sometimes gets confused and gives you a 24−line or 1−line screen, instead of the usual 25 lines? Well, the main culprit is the use of `TERM=vt100` (or some other entry with 24 lines) instead of `TERM=linux` when logged in remotely. If this happens on `/dev/tty2` then typing

```
% cat > /dev/tty2
^[c
^D
```

on some other VT (where 4 symbols are typed to `cat`: ESC, c, ENTER, Ctrl−D) and refreshing the screen on `/dev/tty2` (perhaps using ^L) will fix things. Of course the permanent fix is to use the right termcap or terminfo entry.

Why is it that you sometimes get a lot of line−drawing characters, e.g., after catting a binary to the screen? Well, there are various character set changing escape sequences, and by accident your binary might contain some of these. The ESC c is a general reset, a cure for all, but if you know precisely what went wrong you can repair it without resetting other console attributes. For example, after

```
% cat
^N
^D
```

your shell prompt will be all line−drawing characters. Now do (typing blindly)

```
% cat
^O
^D
```

and all is well again. (Three symbols typed to each `cat`: ^N (or ^O), ENTER, Ctrl−D.) To understand what is happening, see `The console character sets' below.

If you loaded some strange font, and want to return to the default,

```
% setfont
```

will do (provided you stored the default font in the default place). If this default font does not contain an embedded Unicode map (and gives the wrong symbols for accented characters), then say

```
% loadunimap
```

For example, if I do

```
% loadkeys de-latin1
```

then I have a German keyboard, and the key left of the Enter key gives me a−umlaut. This works, because the a−umlaut occurs on the CP437 code page and the kernel Unicode map is initialized to CP437, and my video card has a CP437 font built−in. If I now load an ISO 8859−1 font with

```
% setfont iso01.f16
```

then everything still works, because `setfont` invalidates the kernel Unicode map (if there is no Unicode map attached to the font), and without map the kernel goes directly to the font, and that is precisely correct for an ISO 8859−1 system with `iso01.f16` font. But going back to the previous font with

```
% setfont
```

gives capital Sigma's instead of a−umlaut − all accented letters are mixed up because also this font has no embedded Unicode map. After

```
% loadunimap
```

which loads the default Unicode map (which is right for the default font) all works correctly again. Usually `loadunimap` is not invoked directly, but via `setfont`. Thus, the previous two commands may be replaced by

```
% setfont −u def
```

The Ethiopian fonts and the `lat1u*.psf` fonts have embedded Unicode code map. Most of the others don't.

On old terminals output involving tabs may require a delay, and you have to say

```
% stty tab3
```

(see stty(1)).

You can change the video mode using `resizecons` or `SVGATextMode`. This usually settles the output side. On the input side there are many things that might be wrong. If `X` or `DOOM` or some other program using raw mode crashed, your keyboard may still be in raw (or mediumraw) mode, and it is difficult to give commands. (See "How to get out of raw mode" below.) If you loaded a bad keymap, then

```
% loadkeys -d
```

loads the default map again, but it may well be difficult to type `−'! An alternative is

```
% loadkeys defkeymap
```

Sometimes even the letters are garbled. It is useful to know that there are four main types of keyboards: QWERTY, QWERTZ, AZERTY and DVORAK. The first three are named after the first six letter keys, and roughly represent the English, German and French speaking countries. Compared to QWERTY, the QWERTZ map interchanges Y and Z. Compared to QWERTY, the AZERTY map interchanges Q and A, W and Z, and has its M right of the L, at the semicolon position. DVORAK has an entirely different letter ordering.

4.1 Keyboard hardware reset

Things may be wrong on a lower level than Linux knows about. There are at least two distinct lower levels (keyboard and keyboard controller) where one can give the command "keyboard disable" to the keyboard hardware. Keyboards can often be programmed to use one out of three different sets of scancodes.

However, I do not know of cases where this turned out to be a problem.

Some keyboards have a remapping capability built in. Stormy Henderson (`stormy@Ghost.Net`) writes: `If it's your keyboard accidently being reprogrammed, you can (on a Gateway AnyKey keyboard) press control−alt−suspend_macro to reset the keys to normal.'

5.Delete and Backspace

Getting Delete and Backspace to work just right is nontrivial, especially in a mixed environment, where you talk to console, to `X`, to `bash`, to `emacs`, login remotely, etc. You may have to edit several configuration files to tell all of the programs involved precisely what you want. On the one hand, there is the matter of which keys generate which codes (and how these codes are remapped by e.g. `kermit` or `emacs`), and on the other hand the question of what functions are bound to what codes.

People often complain `my backspace key does not work', as if this key had a built−in function `delete previous character'. Unfortunately, all this key, or any key, does is producing a code, and one only can hope that the kernel tty driver and all application programs can be configured such that the backspace key indeed does function as a `delete previous character' key.

Most Unix programs get their tty input via the kernel tty driver in `cooked' mode, and a simple `stty` command determines the erase character. However, programs like `bash` and `emacs` and `X` do their own input handling, and have to be convinced one–by–one to do the right thing.

5.1 How to tell Unix what character you want to use to delete the last typed character

```
% stty erase ^?
```

If the character is erased, but in a funny way, then something is wrong with your tty settings. If `echoprt` is set, then erased characters are enclosed between `\` and `/`. If `echoe` is not set, then the erase char is echoed (which is reasonable when it is a printing character, like #). Most people will want `stty echoe -echoprt`. Saying `stty sane` will do this and more. Saying `stty -a` shows your current settings. How come this is not right by default? It is, if you use the right `getty`.

Note that many programs (like `bash`, `emacs` etc.) have their own keybindings (defined in `~/.inputrc`, `~/.emacs` etc.) and are unaffected by the setting of the erase character.

The standard Unix tty driver does not recognize a cursor, or keys (like the arrow keys) to move the current position, and hence does not have a command `delete current character' either. But for example you can get `bash` on the console to recognize the Delete key by putting

```
set editing-mode emacs
"\e[3~":delete-char
```

into `~/.inputrc`.

`Getty used to do the right thing with DEL and BS but is broken now?'

Earlier, the console driver would do BS Space BS (`\010\040\010`) when it got a DEL (`\177`). Nowadays, DEL's are ignored (as they should be, since the driver emulates a vt100). Get a better getty, i.e., one that does not output DEL.

`Login behaves differently at the first and second login attempts?'

At the first attempt, you are talking to `getty`. At the second attempt, you are talking to `login`, a different program.

5.2 How to tell Linux what code to generate when a key is pressed

On the console, or, more precisely, when not in (MEDIUM)RAW mode, use

```
% loadkeys mykeys.map
```

and under X use

```
% xmodmap mykeys.xmap
```

Note that (since XFree86−2.1) X reads the Linux settings of the keymaps when initialising the X keymap. Although the two systems are not 100% compatible, this should mean that in many cases the use of `xmodmap` has become superfluous.

For example, suppose that you would like the Backspace key to send a BackSpace (^H, octal 010) and the grey Delete key a DEL (octal 0177). Add the following to `/etc/rc.local` (or wherever you keep your local boot−time stuff):

```
/usr/bin/loadkeys << EOF
keycode 14 = BackSpace
keycode 111 = Delete
EOF
```

Note that this will only change the function of these keys when no modifiers are used. (You need to specify a keymaps line to tell which keymaps should be affected if you want to change bindings on more keymaps.) The Linux kernel default lets Ctrl−Backspace generate BackSpace − this is sometimes useful as emergency escape, when you find you can only generate DELs.

The left Alt key is sometimes called the Meta key, and by default the combinations AltL−X are bound to the symbol MetaX. But what character sequence is MetaX? That is determined (per−tty) by the Meta flag, set by the command `setmetamode`. The two choices are: ESC X or X or−ed with 0200.

`Why doesn't the Backspace key generate BackSpace by default?'

(i) Because the VT100 had a Delete key above the Enter key.

(ii) Because Linus decided so.

5.3 How to tell X to interchange Delete and Backspace

```
% xmodmap -e "keysym BackSpace = Delete" -e "keysym Delete = BackSpace"
```

Or, if you just want the Backspace key to generate a BackSpace:

```
% xmodmap -e "keycode 22 = BackSpace"
```

Or, if you just want the Delete key to generate a Delete:

```
% xmodmap -e "keycode 107 = Delete"
```

(but usually this is the default binding already).

5.4 How to tell emacs what to do when it receives a Delete or Backspace

Put in your `.emacs` file lines like

```
(global-set-key "\?" 'help-command)
(global-set-key "\C-h" 'delete-backward-char)
```

Of course you can bind other commands to other keys in the same way. Note that various major and minor modes redefine keybindings. For example, in incremental search mode one finds the code

```
(define-key map "\177" 'isearch-delete-char)
(define-key map "\C-h" 'isearch-mode-help)
```

This means that it may be a bad idea to use the above two global–set–key commands. There are too many places where there are built–in assumptions about ^H = help and DEL = delete. That doesn't mean that you have to setup keys so that Backspace generates DEL. But if it doesn't then it is easiest to remap them at the lowest possible level in emacs.

5.5 How to tell emacs to interchange Delete and Backspace

Put in your `.emacs` file lines

```
(setq keyboard-translate-table (make-string 128 0))
```

```
(let ((i 0))
  (while (< i 128)
    (aset keyboard-translate-table i i)
    (setq i (1+ i))))
(aset keyboard-translate-table ?\b ?\^?)
(aset keyboard-translate-table ?\^? ?\b)
```

Recent versions of emacs have a function `keyboard-translate` and one may simplify the above to

```
(keyboard-translate ?\C-h ?\C-?)
(keyboard-translate ?\C-? ?\C-h)
```

Note that under X emacs can distinguish between Ctrl–h and the Backspace key (regardless of what codes these produce on the console), and by default emacs will view the Backspace key as DEL (and do deletion things, as bound to that character, rather than help things, bound to ^H). One can distinguish Backspace and Delete, e.g. by

```
(global-unset-key [backspace] )
(global-set-key [backspace] 'delete-backward-char)
(global-unset-key [delete] )
(global-set-key [delete] 'delete-char)
```

5.6 How to tell kermit to interchange Delete and Backspace

Put in your `.kermrc` file the lines

```
set key \127 \8
set key \8 \127
```

5.7 How to tell xterm about your favourite tty modes

Normally xterm will inherit the tty modes from its invoker. Under `xdm`, the default erase and kill characters are `#` and `@`, as in good old Unix Version 6. If you don't like that, you might put something like

```
XTerm*ttymodes: erase ^? kill ^U intr ^C quit ^\ eof ^D \
                susp ^Z start ^Q stop ^S eol ^@
```

in `/usr/lib/X11/app-defaults/XTerm` or in `$HOME/.Xresources`, assuming that you have a line

```
xrdb $HOME/.Xresources
```

in your `$HOME/.xinitrc` or `$HOME/.xsession`.

5.8 How to tell xmosaic that the Backspace key generates a DEL

Putting

```
*XmText.translations: #override\n\
   <Key>osfDelete: delete-previous-character()
*XmTextField.translations: #override\n\
   <Key>osfDelete: delete-previous-character()
```

in your `$HOME/.Xresources` helps.

The netscape FAQ, however, says:

```
Why doesn't my Backspace key work in text fields?
By default, Linux and XFree86 come with the Backspace and Delete keys
misconfigured. All Motif programs (including, of course, Netscape
Navigator) will malfunction in the same way.

The Motif spec says that Backspace is supposed to delete the previous
character and Delete is supposed to delete the following character.
Linux and XFree86 come configured with both the Backspace and Delete
keys generating Delete.

You can fix this by using any one of the xmodmap, xkeycaps, or
loadkeys programs to make the key in question generate the BackSpace
keysym instead of Delete.

You can also fix it by having a .motifbind file; see the man page
for VirtualBindings(3).

Note: Don't use the *XmText.translations or *XmTextField.translations
resources to attempt to fix this problem. If you do, you will blow
away Netscape Navigator's other text-field key bindings.
```

5.9 A better solution for Motif−using programs, like netscape

Ted Kandell (`ted@tcg.net`) suggests the following:

Somewhere in your .profile add the following:

```
stty erase ^H
```

If you are using `bash`, add the following lines to your `.inputrc`:

```
"\C-?": delete-char
"\C-h": backward-delete-char
```

Add the following lines to your .xinitrc file:

```
xmodmap <<-EOF
keycode 22  =  BackSpace osfBackSpace
keycode 107 =  Delete
EOF

# start your window manager here,  for example:
#(fvwm) 2>&1 | tee /dev/tty /dev/console

stty sane
stty erase ^H
loadmap <<-EOF
keycode 14  = BackSpace
keycode 111 = Delete
EOF
```

This will definitely work for a PC 101 or 102 key keyboard with any Linux/XFree86 layout.

The important part to making Motif apps like Netscape work properly is adding osfBackSpace to keycode 22 in addition to BackSpace.

Note that there must be spaces on either side of the = sign.

5.10 What about termcap and terminfo?

When people have problems with backspace, they tend to look at their termcap (or terminfo) entry for the terminal, and indeed, there does exist a kb (or kbs) capability describing the code generated by the Backspace key. However, not many programs use it, so unless you are having problems with one particular program only, probably the fault is elsewhere. Of course it is a good idea anyway to correct your termcap (terminfo) entry. See also below under "The TERM variable".

6.The console character sets

The kernel first tries to figure out what symbol is meant by any given user byte, and next where this symbol is located in the current font.

The kernel knows about 5 translations of bytes into console−screen symbols. In Unicode (UTF−8) mode, the UTF−8 code is just converted directly into Unicode. The assumption is that almost all symbols one needs are present in Unicode, and for the cases where this does not hold the codes 0xff** are reserved for direct font access. When not in Unicode mode, one of four translation tables is used. The four tables are: a) Latin1 −> Unicode, b) VT100 graphics −> Unicode, c) PC −> Unicode, d) user−defined.

There are two character sets, called G0 and G1, and one of them is the current character set. (Initially G0.) Typing ^N causes G1 to become current, ^O causes G0 to become current.

These variables G0 and G1 point at a translation table, and can be changed by the user. Initially they point at tables a) and b), respectively. The sequences ESC (B and ESC (0 and ESC (U and ESC (K cause G0 to point at translation table a), b), c) and d), respectively. The sequences ESC) B and ESC) 0 and ESC) U and ESC) K cause G1 to point at translation table a), b), c) and d), respectively.

The sequence ESC c causes a terminal reset, which is what you want if the screen is all garbled. The oft−advised `echo ^V^O` will only make G0 current, but there is no guarantee that G0 points at table a). In some distributions there is a program reset(1) that just does `echo ^[c`. If your termcap entry for the console is correct (and has an entry `:rs=\Ec:`), then also `setterm -reset` will work.

The user−defined mapping table can be set using mapscrn(8). The result of the mapping is that if a symbol c is printed, the symbol `s = map[c]` is sent to the video memory. The bitmap that corresponds to `s` is found in the character ROM, and can be changed using setfont(8).

7.Console switching

By default, console switching is done using Alt−Fn or Ctrl−Alt−Fn. Under `X` (or recent versions of `dosemu`), only Ctrl−Alt−Fn works. Many keymaps will allow cyclic walks through all allocated consoles using Alt−RightArrow and Alt−LeftArrow.

XFree86 1.3 does not know that Alt is down when you switch to the X window. Thus, you cannot switch immediately to some other VT again but have to release Alt first. In the other direction this should work: the kernel always keeps track of the up/down status of all keys. (As far as possible: on some keyboards some keys do not emit a scancode when pressed (e.g.: the PFn keys of a FOCUS 9000) or released (e.g.: the Pause key of many keyboards).)

XFree86 1.3 saves the fonts loaded in the character ROMs when started, and restores it on a console switch. Thus, the result of `setfont` on a VT is wiped out when you go to X and back. Using `setfont` under X will lead to funny results.

One can change VT under program control using the `chvt` command.

7.1 Changing the number of Virtual Consoles

This question still comes up from time to time, but the answer is: you already have enough of them. Since kernel version 1.1.54, there are between 1 and 63 virtual consoles. A new one is created as soon as it is opened. It is removed by the utility `deallocvt` (but it can be removed only when no processes are associated to it anymore, and no text on it has been selected by programs like `selection` or `gpm`).

For older kernels, change the line

```
#define NR_CONSOLES     8
```

in `include/linux/tty.h` (don't increase this number beyond 63), and recompile the kernel.

If they do not exist yet, create the tty devices with `MAKEDEV` or `mknod ttyN c 4 N` where N denotes the tty number. For example,

```
for i in 9 10 11 12; do mknod /dev/tty$i c 4 $i; done
```

or, better (since it also takes care of owner and permissions),

```
for i in 9 10 11 12; do /dev/MAKEDEV tty$i; done
```

If you want the new VCs to run `getty`, add lines in `/etc/inittab`. (But it is much better to have only two `getty`'s running, and to create more consoles dynamically as the need arises. That way you'll have more memory when you don't use all these consoles, and also more consoles, in case you really need them. Edit `/etc/inittab` and comment out all `getty`'s except for the first two.)

When the consoles are allocated dynamically, it is usually easiest to have only one or two running `getty`. More are opened by `open -l -s bash`. Unused consoles (without associated processes) are deallocated using `deallocvt` (formerly `disalloc`). But, you say, I am involved in activities when I suddenly need more consoles, and do not have a bash prompt available to give the `open` command. Fortunately it is possible to create a new console upon a single keystroke, regardless of what is happening at the current console.

If you have `spawn_login` from `kbd-0.95.tar.gz` and you put

```
loadkeys << EOF
alt keycode 103 = Spawn_Console
EOF
spawn_login &
```

in `/etc/rc.local`, then typing Alt−UpArrow will create a fresh VC running `login` (and switch to it). With `spawn_console &` instead of `spawn_login &` you'll have bash running there. See also `open-1.4.tgz` and `dynamic-vc-1.1.tar.gz`.

What action should be taken upon this Spawn_Console keypress can also be set in `/etc/inittab` under `kbrequest`, if you have a recent `init`. See inittab(5).

(This action can be something entirely different − I just called the key Spawn_Console because that is what I used it for. When used for other purposes it is less confusing to use its synonym KeyboardSignal. For example, some people like to put the lines

```
kb::kbrequest:/sbin/shutdown -h now
```

in `/etc/inittab`, and

```
control alt keycode 79 = KeyboardSignal
control alt keycode 107 = KeyboardSignal
```

in their keymap. Now Ctrl−Alt−End will do a system shutdown.)

You can only login as "root" on terminals listed in `/etc/securetty`. There exist programs that read terminal settings from files `/etc/ttys` and `/etc/ttytype`. If you have such files, and create additional consoles, then it might be a good idea to also add entries for them in these files.

8.Ctrl−Alt−Del and other special key combinations

8.1 Ctrl−Alt−Del (Boot)

If you press Ctrl−Alt−Del (or whatever key was assigned the keysym Boot by loadkeys) then either the machine reboots immediately (without sync), or `init` is sent a SIGINT. The former behaviour is the default. The default can be changed by root, using the system call reboot(), see ctrlaltdel(8). Some `init`'s change the default. What happens when `init` gets SIGINT depends on the version of `init` used − often it will be determined by the pf entry in `/etc/inittab` (which means that you can run an arbitrary program in this case). In the current kernel Ctrl−AltGr−Del is no longer by default assigned to Boot.

8.2 Other combinations

```
Name            Default binding
-------------------------------
Show_Memory     Shift-Scrollock
Show_Registers  AltGr-ScrollLock
Show_State      Ctrl-ScrollLock
Console_n       Alt-Fn and Ctrl-Alt-Fn  (1 <= n <= 12)
Console_{n+12}  AltGr-Fn                (1 <= n <= 12)
Incr_Console    Alt-RightArrow
Decr_Console    Alt-LeftArrow
Last_Console    Alt[Gr]-PrintScreen
Scroll_Backward Shift-PageUp
Scroll_Forward  Shift-PageDown
Caps_On                                 (CapsLock is a toggle; this key sets)
Compose         Ctrl-.
```

8.3 X Combinations

```
Ctrl-Alt-Fn      Switch to VT n
Ctrl-Alt-KP+     Next mode
Ctrl-Alt-KP-     Previous mode
Ctrl-Alt-Backspace      Kill X
```

On some motherboards, Ctrl−Alt−KP− and Ctrl−Alt−KP+ will be equivalent to pressing the Turbo button. That is, both will produce the scancodes 1d 38 4a ca b8 9d and 1d 38 4e ce b8 9d, and both will switch between Turbo (>= 25MHz) and non−Turbo (8 or 12 MHz). (Often these key combinations only function this way when enabled by jumpers on the motherboard.)

Perry F Nguyen (`pfnguyen@netcom22.netcom.com`) writes: AMI BIOS has a feature that locks up the keyboard and flashes the LED's if the Ctrl−Alt−Backspace combination is pressed while a BIOS password is enabled, until the CMOS/BIOS password is typed in.

8.4 Dosemu Combinations

```
Ctrl-Alt-Fn      Switch to VT n (from version 0.50; earlier Alt-Fn)
Ctrl-Alt-PgDn    Kill dosemu (when in RAW keyboard mode)
(and many other combinations - see the dosemu documentation)
```

8.5 Composing symbols

One symbol may be constructed using several keystrokes.

- LeftAlt−press, followed by a decimal number typed on the keypad, followed by LeftAlt−release, yields the symbol with code given by this number. (In Unicode mode this same mechanism, but then with 4 hexadecimal digits, may be used to define a Unicode symbol.)
- A dead diacritic followed by a symbol, yields that symbol adorned with that diacritic. If the combination is undefined, both keys are taken separately. Which keys are dead diacritics is user−settable; none is by default. Five (since 2.0.25 six) dead diacritics can be defined (using loadkeys(1)): dead_grave, dead_acute, dead_circumflex, dead_tilde, dead_diaeresis (and dead_cedilla). Precisely what this adorning means is also user−settable: dead−diacritic, symbol is equivalent to Compose + diacritic + symbol.
- Compose followed by two symbols yields a combination symbol. These combinations are user−settable. Today there are 68 combinations defined by default; you can see them by saying "dumpkeys | grep compose".
- Then there are `Sticky' modifier keys (since 1.3.33). For example, one can type ^C as SControl, C and Ctrl−Alt−BackSpace as SControl, SAlt, BackSpace.

Note that there are at least three such composition mechanisms:

1. The Linux keyboard driver mechanism, used in conjunction with loadkeys.

2. The X mechanism − see X386keybd(1), later XFree86kbd(1). Under X11R6: edit `/usr/X11R6/lib/X11/locale/iso8859-1/Compose`.

   See also Andrew D. Balsa's comments at http://wauug.erols.com/~balsa/linux/deadkeys/index.html.
3. The emacs mechanism obtained by loading "iso−insert.el" or calling `iso−accents−mode'.

For X the order of the two symbols is arbitrary: both Compose−,−c and Compose−c−, yield a c−cedilla; for Linux and emacs only the former sequence works by default. For X the list of compose combinations is fixed. Linux and emacs are flexible. The three default lists are somewhat similar, but the details are different.

8.6 The SysRq key

In case your kernel was compiled with CONFIG_MAGIC_SYSRQ enabled (a feature that is present since Linux 2.1.43) there is a single key (defined in `<linux/keyboard.h>`) to which special system functions are attached, regardless of the current keyboard mode. For the PC architecture this special key is, naturally, the Alt+SysRq key, and any of the two Alt keys will work. (Note that if CONFIG_MAGIC_SYSRQ was not enabled, the default action of this key is to return to the previous console.)

If you press this key, do not release it, and hit another key, a corresponding action is performed. The action is performed whether anybody is logged in or not, is root or not. For the details, see `drivers/char/sysrq.c`. Since this feature is meant only for kernel hackers, that should suffice. Still, let me add a few remarks.

For the key r the keyboard mode is reset to K_XLATE. For the key k a SAK and console reset is done. For the key b the machine is rebooted immediately. (See, not something you want to have enabled on a production machine.) For the key o the power is turned off (when the machine is capable of that). For the key s an emergency sync is scheduled. For the key u an emergency read−only remount is scheduled. For the keys p,t,m various information is shown (namely the same information also shown for RAlt,RCtrl,RShift+ScrollLock). For the keys e,i,l all processes get a SIG_TERM or SIG_KILL, respectively; for l even the init process is killed. Digits set the log level. Anything else prints a short summary: `SysRq: unRaw saK Boot Off Sync Unmount showPc showTasks showMem loglevel0-8 tErm kIll killalL`.

Note: These are very dangerous actions! And they do not use your keymap − indeed, are meant for emergency cases where the state of your keymap, or even of the entire kernel, is uncertain. If you use a dvorak keyboard − bad luck! Most other people will be able to survive: the dangerous letters A,M,Q,W,Y,Z that are differently placed on English, French and German keyboards, are not used for actions.

9.How to get out of raw mode

If some program using K_RAW keyboard mode exits without restoring the keyboard mode to K_XLATE, then it is difficult to do anything − not even Ctrl−Alt−Del works. However, it is sometimes possible to avoid hitting the reset button. (And desirable as well: your users may get angry if you kill their Hack game by rebooting; you might also damage your file system.) Easy solutions involve logging in from another terminal or another machine and doing `kbd_mode -a`. The procedure below assumes that no X is running, that the display is in text mode, and that you are at your bash prompt, that you are using a US keyboard layout, and that your interrupt character is Ctrl−C.

Step 1. Start X. As follows: press 2 (and don't release), press F12 (and don't release) and immediately afterwards press = . This starts X. (Explanation: if a key press produces keycode K, then the key release produces keycode K+128. Probably your shell does not like these high characters, so we avoid generating them by not releasing any key. However, we have to be quick, otherwise key repeat starts. The digit 2 produces a Ctrl−C that discards previous junk, the F12 produces an X and the = a Return.) Probably your screen will be grey now, since no `.xinitrc` was specified. However, Ctrl−Alt−Fn will work and you can go to another VT. (Ctrl−Alt−Backspace also works, but that exits X, and gets you back into the previous state, which is not what you want.)

Step 2. Setup to change the keyboard mode. (For example, by `sleep 5; kbd_mode -a`.)

Step 3. Leave X again. Alt−Fx (often Alt−F7) brings you back to X, and then Ctrl−Alt−Backspace exits X. Within 5 seconds your keyboard will be usable again.

If you want to prepare for the occasion, then make `\215A\301` (3 symbols) an alias for `kbd_mode -a`. Now just hitting = F7 = (3 symbols) will return you to sanity.

10.The keyboard LEDs

1. There are per−tty keyboard flags: each VC has its own NumLock, CapsLock, ScrollLock. By default these keyboard flags are shown in the LEDs. The usual way to change them is by pressing the corresponding key. (Side remark: pressing the NumLock key when in application key mode will not change the NumLock status, but produce an escape sequence. If you want the NumLock key to always change the Numlock status, bind it to Bare_Num_Lock.)

2. Next, there are per−tty default keyboard flags, to initialize the keyboard flags when a reset occurs. Thus if you want NumLock on all the time, that is possible. The usual way to change them is by \``setleds -D` ...'.

3. There is the possibility that the leds do not reflect the keyboard flags, but something else.

3A. This something else can be three bits somewhere in the kernel − which can be used if you want to monitor some hardware or software status bit(s). If you want this, edit the kernel source to call `register_leds()` somewhere.

3B. This something else can also be whatever some user program wants to show in the LEDs. Thus, people who like such things can make nice patterns of lights. If you want this, use the KDSETLED ioctl.

This latter use is not per−tty, but the choice between former and latter use is per−tty.

Summarizing: Each tty has a flag `kbd->ledmode`. If this has the value LED_SHOW_FLAGS then the keyboard flags (NumLock etc.) of that tty are shown. If this has the value LED_SHOW_MEM then three selected memory addresses are shown. If this has the value LED_SHOW_IOCTL then the leds show whatever value was last assigned to them using the KDSETLED ioctl.

One may add that X uses ioctl's to set the LEDs, but fails to reset its VT when it exits, so after using X there may be one VT that is not in the default LED_SHOW_FLAGS state. This can be fixed by doing `` `setleds -L' `` on that VT. See setleds(1).

11.The TERM variable

Many programs use the `TERM` variable and the database `/etc/termcap` or `/usr/lib/terminfo/*` to decide which strings to send for clear screen, move cursor, etc., and sometimes also to decide which string is sent by the users backspace key, function keys etc. This value is first set by the kernel (for the console). Usually, this variable is re−set by `getty`, using `/etc/ttytype` or the argument specified in `/etc/inittab`. Sometimes, it is also set in `/etc/profile`.

Older systems use `TERM=console` or `TERM=con80x25`. Newer systems (with ncurses 1.8.6) use the more specific `TERM=linux` or `TERM=linux-80x25`. However, old versions of `setterm` test for `TERM=con*` and hence fail to work with `TERM=linux`.

Since kernel version 1.3.2, the kernel default for the console is `TERM=linux`.

If you have a termcap without entry for linux, add the word linux to the entry for the console:

```
        console|con80x25|linux:\
```

and make `/usr/lib/terminfo/l/linux` a copy of or symbolic link to `/usr/lib/terminfo/c/console`.

11.1 Terminfo

The terminfo entry for the linux console from ncurses 1.8.6 misses the entry `kich1=\E[2~`, needed by some programs. Edit the file and `tic` it.

12.How to make other programs work with non−ASCII chars

In the bad old days this used to be quite a hassle. Every separate program had to be convinced individually to leave your bits alone. Not that all is easy now, but recently a lot of gnu utilities have learned to react to `LC_CTYPE=iso_8859_1` or `LC_CTYPE=iso-8859-1`. Try this first, and if it doesn't help look at the hints below. Note that in recent versions of libc the routine setlocale() only works if you have installed the locale files (e.g. in `/usr/lib/locale`).

First of all, the 8−th bit should survive the kernel input processing, so make sure to have `stty cs8 -istrip -parenb` set.

A. For `emacs` the details strongly depend on the version. The information below is for version 19.34. Put lines

```
(set-input-mode nil nil 1)
(standard-display-european t)
(require 'iso-syntax)
```

into your `$HOME/.emacs`. The first line (to be precise: the final 1) tells `emacs` not to discard the 8−th bit from input characters. The second line tells `emacs` not to display non−ASCII characters as octal escapes. The third line specifies the syntactic properties and case conversion table for the Latin−1 character set These last two lines are superfluous if you have something like `LC_CTYPE=ISO-8859-1` in your environment. (The variable may also be `LC_ALL` or even `LANG`. The value may be anything with a substring `88591' or `8859−1' or `8859_1'.)

This is a good start. On a terminal that cannot display non−ASCII ISO 8859−1 symbols, the command

```
(load-library "iso-ascii")
```

will cause accented characters to be displayed comme {,c}a. If your keymap does not make it easy to produce non−ASCII characters, then

```
(load-library "iso-transl")
```

will make the 2−character sequence Ctrl−X 8 a compose character, so that the 4−character sequence Ctrl−X 8 , c produces c−cedilla. Very inconvenient.

The command

```
(iso-accents-mode)
```

will toggle ISO−8859−1 accent mode, in which the six characters ', `, ", ^, ~, / are dead keys modifying the following symbol. Special combinations: ~c gives a c with cedilla, ~d gives an Icelandic eth, ~t gives an Icelandic thorn, "s gives German sharp s, /a gives a with ring, /e gives an a−e ligature, ~< and ~> give guillemots, ~! gives an inverted exclamation mark, ~? gives an inverted question mark, and '' gives an acute accent. This is the default mapping of accents. The variable `iso-languages` is a list of pairs (language name, accent mapping), and a non−default mapping can be selected using

```
(iso-accents-customize LANGUAGE)
```

Here LANGUAGE can be one of `"portuguese"`, `"irish"`, `"french"`, `"latin-2"`, `"latin-1"`.

Since the Linux default compose character is Ctrl−. it might be convenient to use that everywhere. Try

```
(load-library "iso-insert.el")
(define-key global-map [?\C-.] 8859-1-map)
```

The latter line will not work under xterm, if you use emacs -nw, but in that case you can put

```
XTerm*VT100.Translations:       #override\n\
      Ctrl <KeyPress> . : string("\0308")
```

in your .Xresources.)

B. For less, put LESSCHARSET=latin1 in the environment. This is also what you need if you see \255 or <AD> in man output: some versions of less will render the soft hyphen (octal 0255, hex 0xAD) this way when not given permission to output Latin-1.

C. For ls, give the option -N. (Probably you want to make an alias.)

D. For bash (version 1.13.*), put

```
set meta-flag on
set convert-meta off
```

and, according to the Danish HOWTO,

```
set output-meta on
```

into your $HOME/.inputrc.

E. For tcsh, use

```
setenv LANG     US_en
setenv LC_CTYPE iso_8859_1
```

If you have nls on your system, then the corresponding routines are used. Otherwise tcsh will assume iso_8859_1, regardless of the values given to LANG and LC_CTYPE. See the section NATIVE LANGUAGE SYSTEM in tcsh(1). (The Danish HOWTO says: setenv LC_CTYPE ISO-8859-1; stty pass8)

F. For flex, give the option -8 if the parser it generates must be able to handle 8-bit input. (Of course it must.)

G. For elm, set displaycharset to ISO-8859-1. (Danish HOWTO: LANG=C and LC_CTYPE=ISO-8859-1)

H. For programs using curses (such as lynx) David Sibley reports: The regular curses package uses the high-order bit for reverse video mode (see flag _STANDOUT defined in /usr/include/curses.h). However, ncurses seems to be 8-bit clean and does display iso-latin-8859-1 correctly.

I. For programs using groff (such as man), make sure to use -Tlatin1 instead of -Tascii. Old versions of the program man also use col, and the next point also applies.

J. For `col`, make sure 1) that it is fixed so as to do `setlocale(LC_CTYPE,"");` and 2) put `LC_CTYPE=ISO-8859-1` in the environment.

K. For `rlogin`, use option `-8`.

L. For `joe`, `sunsite.unc.edu:/pub/Linux/apps/editors/joe-1.0.8-linux.tar.gz` is said to work after editing the configuration file. Someone else said: `joe`: Put the `-asis` option in `/isr/lib/joerc` in the first column.

M. For LaTeX: `\documentstyle[isolatin]{article}`. For LaTeX2e: `\documentclass{article}\usepackage{isolatin}` where `isolatin.sty` is available from ftp://ftp.vlsivie.tuwien.ac.at/pub/8bit.

A nice discussion on the topic of ISO−8859−1 and how to manage 8−bit characters is contained in the file `grasp.insa-lyon.fr:/pub/faq/fr/accents` (in French). Another fine discussion (in English) can be found in ftp.vlsivie.tuwien.ac.at:/pub/8bit/FAQ−ISO−8859−1, which is mirrored in rtfm.mit.edu:pub/usenet−by−group/comp.answers/character−sets/iso−8859−1−faq.

If you need to fix a program that behaves badly with 8−bit characters, one thing to keep in mind is that if you have a signed char type then characters may be negative, and using them as an array index will fail. Several programs can be fixed by judiciously adding (unsigned char) casts.

13.What precisely does XFree86−2.1 do when it initializes its keymap?

Since version 2.1, XFree86 will initialize its keymap from the Linux keymap, as far as possible. However, Linux had 16 entries per key (one for each combination of the Shift, AltGr, Ctrl, Alt modifiers) and presently has 256 entries per key, while X has 4 entries per key (one for each combination of Shift, Mod), so some information is necessarily lost.

First `X` reads the `Xconfig` file, where definitions of the LeftAlt, RightAlt, RightCtl, ScrollLock keys as Meta, ModeShift, Compose, ModeLock or ScrollLock might be found − see X386keybd(1), later XFree86kbd(1).

For Mod the LeftAlt key is taken, unless RightCtl was defined as ModeShift or ModeLock, in which case RightCtl is taken, or RightAlt was so defined, in which case RightAlt is taken. This determines how the 4 XFree86 meanings of a key are selected from the 16 Linux meanings. Note that Linux today does not distinguish by default between the two Ctrl keys or between the two Shift keys. `X` does distinguish.

Now the kernel keymap is read and the usually obvious corresponding X bindings are made. The bindings for the "action keys" Show_Memory, Show_State, Show_Registers, Last_Console, Console_n, Scroll_Backward, Scroll_Forward, Caps_On and Boot are ignored, as are the dead diacriticals, and the locks (except for ShiftLock), and the "ASCII−x" keys.

Next, the definitions in the `Xconfig` file are used. (Thus, a definition of Compose in `Xconfig` will override its value as found in the Linux keymap.)

What happens to the strings associated with the function keys? Nothing, X does not have such a concept. (But it is possible to define strings for function keys in `xterm` – note however that the window manager gets the keys first.)

I don't know how to convince `xterm` that it should use the X keymap when Alt is pressed; it seems just to look at its resource `eightBitInput`, and depending on whether that is true or false either set the high order bit of the character, or generate an additional Escape character (just like setmetamode(1) does for the console).

14.Unusual keys and keyboards

The two keys PrintScrn/SysRq and Pause/Break are special in that they have two keycodes: the former has keycode 84 when Alt is pressed simultaneously, and keycode 99 otherwise; the latter has keycode 101 when Ctrl is pressed simultaneously, and keycode 119 otherwise. (Thus, it makes no sense to bind functions to Alt keycode 99 or Ctrl keycode 119.)

If you have strange keys, that do not generate any code under Linux (or generate messages like "unrecognized scancode"), and your kernel is 1.1.63 or later, then you can use setkeycodes(1) to tell the kernel about them. They won't work under X, however. Once they have gotten a keycode from `setkeycodes`, they can be assigned a function by `loadkeys`.

15.Examples of use of loadkeys and xmodmap

Switching Caps Lock and Control on the keyboard (assuming you use keymaps 0–15; check with `dumpkeys | head -1`)

```
% loadkeys
keymaps 0-15
keycode 58 = Control
keycode 29 = Caps_Lock
%
```

Switching them under X only:

```
% xmodmap .xmodmaprc
```

where `.xmodmaprc` contains lines

```
remove Lock = Caps_Lock
remove Control = Control_L
keysym Control_L = Caps_Lock
keysym Caps_Lock = Control_L
```

```
add Lock = Caps_Lock
add Control = Control_L
```

What is this about the key numbering? Backspace is 14 under Linux, 22 under X? Well, the numbering can best be regarded as arbitrary; the Linux number of a key can be found using showkey(1), and the X number using xev(1). Often the X number will be 8 more than the Linux number.

Something else people like to change are the bindings of the function keys. Suppose that you want to make F12 produce the string "emacs ". Then

```
% loadkeys
keycode 88 = F12
string F12 = "emacs "
%
```

will do this. More explicitly, the procedure is like this: (i) find the keycodes of the keys to be remapped, using showkey(1). (ii) save the current keymap, make a copy and edit that:

```
% dumpkeys > my_keymap
% cp my_keymap trial_keymap
% emacs trial_keymap
% loadkeys trial_keymap
%
```

The format of the table can be guessed by looking at the output of `dumpkeys`, and is documented in keytables(5). When the new keymap functions as desired, you can put an invocation

```
loadkeys my_new_keymap
```

in `/etc/rc.local` or so, to execute it automatically at boot–up. Note that changing modifier keys is tricky, and a newbie can easily get into a situation only an expert can get out of.

The default directory for keymaps is `/usr/lib/kbd/keytables`. The default extension for keymaps is `.map`. Thus, `loadkeys uk` would probably load `/usr/lib/kbd/keytables/uk.map`.

(On my machine) `/dev/console` is a symbolic link to `/dev/tty0`, and the kernel regards `/dev/tty0` as a synonym for the current VT. XFree86 1.3 changes the owner of `/dev/tty0`, but does not reset this after finishing. Thus, `loadkeys` or `dumpkeys` might fail because someone else owns `/dev/tty0`; in such a case you might run X first. Note that you cannot change keyboard mappings when not at the console (and not superuser).

15.1 `I can use only one finger to type with'

"Can the Shift, Ctrl and Alt keys be made to behave as toggles?"

Yes, after saying

```
% loadkeys
keymaps 0-15
keycode 29 = Control_Lock
keycode 42 = Shift_Lock
keycode 56 = Alt_Lock
%
```

the left Control, Shift and Alt keys will act as toggles. The numbers involved are revealed by showkey (and usually are 29, 97, 42, 54, 56, 100 for left and right control, shift and alt, respectively), and the functions are Control_Lock, Shift_Lock, Alt_Lock, ALtGr_Lock.

"What about `sticky' modifier keys?"

Since version 1.3.33, the kernel knows about `sticky' modifier keys. These act on the next key pressed. So, where one earlier needed the 3−symbol sequence Shift_Lock a Shift_Lock to type `A', one can now use the 2−symbol sequence SShift_Lock a. Versions of the kbd package older than 0.93 do not yet include code for these sticky modifiers, and have to invoke them using their hexadecimal codes. For example,

```
% loadkeys
keymaps 0-15
keycode 54 = 0x0c00
keycode 97 = 0x0c02
keycode 100 = 0x0c03
%
```

will make the right Shift, Ctrl, Alt sticky versions of the left ones. >From 0.93 on you can say

```
% loadkeys
keymaps 0-15
keycode 54 = SShift
keycode 97 = SCtrl
keycode 100 = SAlt
%
```

to obtain the same result. This will allow you to type Ctrl−Alt−Del in three keystrokes with one hand.

The keymaps line in these examples should cover all keymaps you have in use. You find what keymaps you have in use by

```
% dumpkeys | head -1
```

16.Changing the video mode

As far as I know there are 6 ways to change resolution:

1. At compile time: change the line

```
SVGA_MODE=      -DSVGA_MODE=NORMAL_VGA
```

in `/usr/src/linux/Makefile`.

1A. After compilation: use `rdev -v` – a terrible hack, but it exists.

2. At boot time: put `vga=ask` in the lilo config file, and lilo will ask you what video mode you want. Once you know, put `vga=`mypreference.

3. At run time: A. Use the `resizecons` command. (This is a very primitive wrapper around the VT_RESIZE ioctl.) B. Use the `SVGATextMode` command. (This is a less primitive wrapper around the VT_RESIZE ioctl.)

4. Not "on the console": Under `dosemu`, or with svgalib etc. you can change the hardware video mode without the console driver being aware of it. Sometimes this is useful in getting `resizecons` or `SVGATextMode` set up: use `dosemu` and some DOS program to get into the desired videomode, dump (say from another VT) the contents of all video hardware registers, and use that in the initialization that `resizecons` and `SVGATextMode` require. In some cases where the video mode has gotten into some unusable state, starting `dosemu`, relying on the BIOS to set up the video mode, and then killing `dosemu` (with `kill -9`), is the easiest way to get into shape again.

16.1 Instructions for the use of resizecons

Get svgalib and compile the program `restoretextmode`. Boot up your machine in all possible video modes (using `vga=ask` in the lilo config file), and write the video hardware register contents to files CxR (C=cols, R=rows), e.g., 80x25, 132x44, etc. Put these files in `/usr/lib/kbd/videomodes`. Now `resizecons 132x44` will change videomode for you (and send SIGWINCH to all processes that need to know about this, and load another font if necessary).

At present, `resizecons` only succeeds when there is memory enough for both the old and the new consoles at the same time.

17.Changing the keyboard repeat rate

At startup, the Linux kernel sets the repeat rate to its maximal value. For most keyboards this is reasonable, but for some it means that you can hardly touch a key without getting three copies of the corresponding symbol. Use the program kbdrate(8) to change the repeat rate, or, if that doesn't help, edit or remove the section

```
! set the keyboard repeat rate to the max

    mov     ax,#0x0305
    xor     bx,bx           ! clear bx
    int     0x16
```

of `/usr/src/linux/[arch/i386/]boot/setup.S`.

18.Scrolling

There are two ways to get a screen to scroll. The first, called `hard scrolling', is to leave the text in video memory as it is, but change the viewing origin. This is very fast. The second, called `soft scrolling', involves moving all screen text up or down. This is much slower. The kernel console driver will write text starting at the top of the video memory, continuing to the bottom, then copy the bottom part to the top again, and continue, all the time using hard scrolling to show the right part on the screen. You can scroll back until the top op the video memory by using Shift−PageUp (the grey PageUp) and scroll down again using Shift−PageDown (the grey PageDown), assuming a default keymap. The amount of scrollback is thus limited to the amount of video memory you happen to have and you cannot increase this amount. If you need more scrollback, use some program that buffers the text, like `less` or `screen` − by using a buffer on disk you can go back to what you did last week. (One can set the amount of scrollback for `xterm` by adding a line like `XTerm*saveLines: 2500` in `.Xresources`.)

Upon changing virtual consoles, the screen content of the old VT is copied to kernel memory, and the screen content of the new VT is copied from kernel memory to video memory. Only the visible screen is copied, not all of video memory, so switching consoles means losing the scrollback information.

Sometimes, hard scrolling is undesirable, for example when the hardware does not have the possibility to change viewing origin. The first example was a Braille machine that would render the top of video memory in Braille. There is a kernel boot−time option `no−scroll` to tell the console driver not to use hard scrolling. See bootparam(7).

19.Screensaving

`setterm -blank`*nn* will tell the console driver to blank the screen after *nn* minutes of inactivity. (With *nn* = 0, screensaving is turned off. In some old kernels this first took effect after the next keyboard interrupt.)

The `s` option of xset(1) will set the X screensaving parameters: `xset s off` turns off the screensaver, `xset s 10` blanks the screen after 10 minutes.

The video hardware powersaving modes can be enabled/disabled using the `setvesablank` program given in the starting comment of `/usr/src/linux/drivers/char/vesa_blank.c`.

20.Screen dumps

`setterm -dump`*N* will dump the contents of the screen of `/dev/tty`*N* to a file `screen.dump` in the current directory. See setterm(1).

The current contents of the screen of `/dev/tty`*N* can be accessed using the device `/dev/vcs`*N* (where `vcs' stands for `virtual console screen'). For example, you could have a clock program that displays the current time in the upper right hand corner of the console screen (see the program `vcstime` in `kbd-0.95.tar.gz`). Just dumping the contents goes with `cat /dev/vcs`*N*. These device files `/dev/vcs`*N* do not contain newlines, and do not contain attributes, like colors. From a program it is usually better to use `/dev/vcsa`*N* (`virtual console screen with attributes') instead -- it starts with a header giving the number of rows and columns and the location of the cursor. See vcs(4).

21.Some properties of the VT100 -- application key mode

: Sometimes my cursor keys or keypad keys produce strange codes?

When the terminal is in application cursor key mode the cursor keys produce Esc O x and otherwise Esc [x where x is one of A,B,C,D. Certain programs put the terminal in application cursor key mode; if you kill them with `kill -9`, or if they crash, then the mode will not be reset.

```
% echo -e '\033c'
```

resets all properties of the current VC. Just changing the cursor application key mode is done by

```
% echo -e '\033[?1h'
```

(set) and

```
% echo -e '\033[?1l'
```

(clear).

When the terminal is in application keypad key mode the keypad keys produce Esc O y and otherwise Esc [z ~ for certain y and z. Setting application keypad key mode is done by

```
% echo -e '\033='
```

and

```
% echo -e '\033>'
```

clears it again.

22.Hardware incompatibility

Several people have noticed that they lose typed characters when a floppy disk is active. It seems that this might be a problem with Uni−486WB motherboards. (Please mail me (`aeb@cwi.nl`) to confirm [yes, I have the same problem], deny [no, nothing wrong with my Uni−486WB], modify [My Xyzzy machine has the same problem].)

Tjalling Tjalkens (`tjalling@ei.ele.tue.nl`) reports very similar problems with "a no−brand GMB−486 UNP Vesa motherboard with AMD 486DX2−66 CPU" − during floppy activity some keystrokes are lost, during floppy tape streamer (Conner C 250 MQ) activity many keystrokes are lost.

Some people experience sporadic lockups − sometimes associated to hard disk activity or other I/O.

Ulf Tietz (`ulf@rio70.bln.sni.de`) wrote: `I have had the same problems, when I had my motherboard tuned too fast. So I reset all the timings (CLK, wait statements etc) to more conventional values, and the problems are gone.'

Bill Hogan (`bhogan@crl.com`) wrote: `If you have an AMI BIOS, you might try setting the Gate A20 emulation parameter to "chipset" (if you have that option). Whenever I have had that parameter set to any of the other options on my machine ("fast", "both", "disabled") I have had frequent keyboard lockups.'

23.Copyright

Additions and corrections are welcome. Andries Brouwer − `aeb@cwi.nl`
