NAME

keymaps – keyboard table descriptions for loadkeys and dumpkeys

DESCRIPTION

These files are used by **loadkeys**(1) to modify the translation tables used by the kernel keyboard driver and generated by **dumpkeys**(1) from those translation tables.

The format of these files is vaguely similar to the one accepted by **xmodmap**(1). The file consists of charset or key or string definition lines interspersed with comments.

Comments are introduced with **!** or **#** characters and continue to the end of the line. Anything following one of these characters on that line is ignored. Note that comments need not begin from column one as with **xmodmap**(1).

The syntax of keymap files is line oriented; a complete definition must fit on a single logical line. Logical lines can, however, be split into multiple physical lines by ending each subline with the backslash character (\).

INCLUDE FILES

A keymap can include other keymaps using the syntax

include "pathname"

CHARSET DEFINITIONS

A character set definition line is of the form:

```
charset "iso-8859-x"
```

It defines how following keysyms are to be interpreted. For example, in iso-8859-1 the symbol mu (or micro) has code 0265, while in iso-8859-7 the letter mu has code 0354.

COMPLETE KEYCODE DEFINITIONS

Each complete key definition line is of the form:

keycode *keynumber* **=** *keysym keysym keysym*...

keynumber is the internal identification number of the key, roughly equivalent to the scan code of it. *keynumber* can be given in decimal, octal or hexadecimal notation. Octal is denoted by a leading zero and hexadecimal by the prefix **0x.**

Each of the *keysyms* represent keyboard actions, of which up to 256 can be bound to a single key. The actions available include outputting character codes or character sequences, switching consoles or keymaps, booting the machine etc. (The complete list can be obtained from dumpkeys(1) by saying **dumpkeys -l** .)

Each *keysym* may be prefixed by a '+' (plus sign), in wich case this keysym is treated as a "letter" and therefore affected by the "CapsLock" the same way as by "Shift" (to be correct, the CapsLock inverts the Shift state). The ASCII letters ('a'-'z' and 'A'-'Z') are made CapsLock'able by default. If Shift+CapsLock should not produce a lower case symbol, put lines like

keycode 30 = +a A

in the map file.

Which of the actions bound to a given key is taken when it is pressed depends on what modifiers are in effect at that moment. The keyboard driver supports 9 modifiers. These modifiers are labeled (completely arbitrarily) Shift, AltGr, Control, Alt, ShiftL, ShiftR, CtrlL, CtrlR and CapsShift. Each of these modifiers has an associated weight of power of two according to the following table:

modifier	*weight*
Shift	1
AltGr	2
Control	4
Alt	8
ShiftL	16

ShiftR	32
CtrlL	64
CtrlR	128
CapsShift	256

The effective action of a key is found out by adding up the weights of all the modifiers in effect. By default, no modifiers are in effect, so action number zero, i.e. the one in the first column in a key definition line, is taken when the key is pressed or released. When e.g. Shift and Alt modifiers are in effect, action number nine (from the 10th column) is the effective one.

Changing the state of what modifiers are in effect can be achieved by binding appropriate key actions to desired keys. For example, binding the symbol Shift to a key sets the Shift modifier in effect when that key is pressed and cancels the effect of that modifier when the key is released. Binding AltGr_Lock to a key sets AltGr in effect when the key is pressed and cancels the effect when the key is pressed again. (By default Shift, AltGr, Control and Alt are bound to the keys that bear a similar label; AltGr may denote the right Alt key.)

Note that you should be very careful when binding the modifier keys, otherwise you can end up with an unusable keyboard mapping. If you for example define a key to have Control in its first column and leave the rest of the columns to be VoidSymbols, you're in trouble. This is because pressing the key puts Control modifier in effect and the following actions are looked up from the fifth column (see the table above). So, when you release the key, the action from the fifth column is taken. It has VoidSymbol in it, so nothing happens. This means that the Control modifier is still in effect, although you have released the key. Re-pressing and releasing the key has no effect. To avoid this, you should always define all the columns to have the same modifier symbol. There is a handy short-hand notation for this, see below.

keysyms can be given in decimal, octal, hexadecimal, unicode or symbolic notation. The numeric notations use the same format as with *keynumber*. Unicode notation is "U+" followed by four hexadecimal digits. The symbolic notation resembles that used by **xmodmap**(1). Notable differences are the number symbols. The numeric symbols '0', ..., '9' of **xmodmap**(1) are replaced with the corresponding words 'zero', 'one', ... 'nine' to avoid confusion with the numeric notation.

It should be noted that using numeric notation for the *keysyms* is highly unportable as the key action numbers may vary from one kernel version to another and the use of numeric notations is thus strongly discouraged. They are intended to be used only when you know there is a supported keyboard action in your kernel for which your current version of **loadkeys**(1) has no symbolic name.

There is a number of short-hand notations to add readability and reduce typing work and the probability of typing-errors.

First of all, you can give a map specification line, of the form

```
keymaps 0-2,4-5,8,12
```

to indicate that the lines of the keymap will not specify all 256 columns, but only the indicated ones. (In the example: only the plain, Shift, AltGr, Control, Control+Shift, Alt and Control+Alt maps, that is, 7 columns instead of 256.) When no such line is given, the keymaps 0-M will be defined, where M+1 is the maximum number of entries found in any definition line.

Next, you can leave off any trailing VoidSymbol entries from a key definition line. VoidSymbol denotes a keyboard action which produces no output and has no other effects either. For example, to define key number 30 to output 'a' unshifted, 'A' when pressed with Shift and do nothing when pressed with AltGr or other modifiers, you can write

```
keycode  30 = a    A
```

instead of the more verbose

```
keycode  30 = a    A          VoidSymbol        VoidSymbol \
                   VoidSymbol VoidSymbol VoidSymbol ...
```

For added convenience, you can usually get off with still more terse definitions. If you enter a key definition line with only and exactly one action code after the equals sign, it has a special meaning. If the code

(numeric or symbolic) is not an ASCII letter, it means the code is implicitly replicated through all columns being defined. If, on the other hand, the action code is an ASCII character in the range 'a', ..., 'z' or 'A', ..., 'Z' in the ASCII collating sequence, the following definitions are made for the different modifier combinations, provided these are actually being defined. (The table lists the two possible cases: either the single action code is a lower case letter, denoted by 'x' or an upper case letter, denoted by 'Y'.)

modifier	*symbol*	
none	x	Y
Shift	X	y
AltGr	x	Y
Shift+AltGr	X	y
Control	Control_x	Control_y
Shift+Control	Control_x	Control_y
AltGr+Control	Control_x	Control_y
Shift+AltGr+Control	Control_x	Control_y
Alt	Meta_x	Meta_Y
Shift+Alt	Meta_X	Meta_y
AltGr+Alt	Meta_x	Meta_Y
Shift+AltGr+Alt	Meta_X	Meta_y
Control+Alt	Meta_Control_x	Meta_Control_y
Shift+Control+Alt	Meta_Control_x	Meta_Control_y
AltGr+Control+Alt	Meta_Control_x	Meta_Control_y
Shift+AltGr+Control+Alt	Meta_Control_x	Meta_Control_y

SINGLE MODIFIER DEFINITIONS

All the previous forms of key definition lines always define all the M+1 possible modifier combinations being defined, whether the line actually contains that many action codes or not. There is, however, a variation of the definition syntax for defining only single actions to a particular modifier combination of a key. This is especially useful, if you load a keymap which doesn't match your needs in only some modifier combinations, like AltGr+function keys. You can then make a small local file redefining only those modifier combinations and loading it after the main file. The syntax of this form is:

{ **plain** | <modifier sequence> } **keycode** *keynumber* **=** *keysym*

, e.g.,

```
plain keycode 14 = BackSpace
control alt keycode 83 = Boot
alt keycode 105 = Decr_Console
alt keycode 106 = Incr_Console
```

Using "plain" will define only the base entry of a key (i.e. the one with no modifiers in effect) without affecting the bindings of other modifier combinations of that key.

STRING DEFINITIONS

In addition to comments and key definition lines, a keymap can contain string definitions. These are used to define what each function key action code sends. The syntax of string definitions is:

string *keysym* **=** "text"

text can contain literal characters, octal character codes in the format of backslash followed by up to three octal digits, and the three escape sequences **\n**, ****, and **\"**, for newline, backslash and quote, respectively.

COMPOSE DEFINITIONS

Then there may also be compose definitions. They have syntax

compose '*char***' '***char***' to '***char***'**

and describe how two bytes are combined to form a third one (when a dead accent or compose key is used). This is used to get accented letters and the like on a standard keyboard.

ABBREVIATIONS

Various abbreviations can be used with kbd-0.96 and later.

strings as usual
Defines the usual values of the strings (but not the keys they are bound to).

compose as usual for "iso-8859-1"
Defines the usual compose combinations.

To find out what *keysyms* there are available for use in keymaps, use the command

dumpkeys --long-info

Unfortunately, there is currently no description of what each symbol does. It has to be guessed from the name or figured out from the kernel sources.

EXAMPLES

(Be careful to use a keymaps line, like the first line of ‘dumpkeys‘, or "keymaps 0-15" or so.)

The following entry exchanges the left Control key and the Caps Lock key on the keyboard:

```
keycode  58 = Control
keycode  29 = Caps_Lock
```

Key number 58 is normally the Caps Lock key, and key number 29 is normally the Control key.

The following entry sets the Shift and Caps Lock keys to behave more nicely, like in older typewriters. That is, pressing Caps Lock key once or more sets the keyboard in CapsLock state and pressing either of the Shift keys releases it.

```
keycode  42 = Uncaps_Shift
keycode  54 = Uncaps_Shift
keycode  58 = Caps_On
```

The following entry sets the layout of the edit pad in the enhanced keyboard to be more like that in the VT200 series terminals:

```
keycode 102 = Insert
keycode 104 = Remove
keycode 107 = Prior
shift keycode 107 = Scroll_Backward
keycode 110 = Find
keycode 111 = Select
control alt   keycode 111 = Boot
control altgr keycode 111 = Boot
```

Here’s an example to bind the string "du\ndf\n" to the key AltGr-D. We use the "spare" action code F100 not normally bound to any key.

```
altgr keycode 32 = F100
string F100 = "du\ndf\n"
```

SEE ALSO

loadkeys(1), **dumpkeys**(1), **showkey**(1), **xmodmap**(1)