

# Linux: Unbuffered reading from stdin

---

*Source:* [http://coding.derkeiler.com/Archive/C\\_CPP/comp.lang.c/2007-10/msg03033.html](http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2007-10/msg03033.html)

---

- *From:* Markus Mayer <[contact.me.via.my.website@xxxxxxx](mailto:contact.me.via.my.website@xxxxxxx)>
  - *Date:* Mon, 22 Oct 2007 13:24:49 +0200
- 

Hi folks.

I'm somewhat new to \*nix programming and just ran into a problem. I have to take user input from the terminal but like to constrain that to some rules given, i.e. "numbers only" or "alphanumeric only" etc.

scanf would do, but I don't like the fact that wrong ('disallowed') characters are visible and all that stuff. For that I need unbuffered input. Since there seems to be no atomic function to do that I tried to fiddle around with ioctl() and that seems to work fine. Currently I flag out ICANON and ECHO from the tty descriptor, set a minimum input of 1 byte, getch() or read() a byte, check, printf(), fine.

However things become nasty when the user presses an key that sends an escape sequence. I've seen some code on the web that does the following approach: Check for character==27. If so, read() 4 bytes into a buffer, zero terminate the buffer accordingly then strcmp() the buffer against the actual escape sequences.

But then there are two problems:

1) The escape sequences seem to change with every terminal type. On my terminal the codes for F1 to F5 are ^OP, ^OQ, ^OR, ^OS, ^[15~ (etc) while on the code given in the web they used to be [[A, [[B etc. Are there some defines?

2) What happens when the user presses just ESC? Sounds easy to me, just check if there are no more chars in stdin – but how do you do that? By just read()ing STDIN\_FILENO after an ESC the terminal is blocked until another key is pressed – that is not really what I want.

Any ideas? Or am I reinventing the wheel?

Regards,  
Markus

.