http://invisible-island.net/ncurses/
Thomas E. Dickey
$Date: 2014/12/13 14:33:57 $

Here is the latest version of this file.

What is NCURSES?

Ncurses (new curses, pronounced "*enn*-curses") started as a freely distributable "clone" of System V Release 4.0 (SVr4) curses. It has outgrown the "clone" description, and now contains many features which are not in SVr4 curses. Curses is a pun on the term "cursor optimization". It is a library of functions that manage an application's display on character-cell terminals (e.g., VT100).

The name "ncurses" was first used as the name of the curses library in Pavel Curtis's `pcurses`, dated 1982. It was apparently developed on a BSD 4.4 system, at Cornell. Parts of pcurses are readily identifiable in ncurses, including the basics for the terminfo compiler (named `compile` in that package):

- the `Caps`, used to define the terminfo capabilities
- awk scripts `MKcaptab.awk`, `MKnames.awk`
- the library modules used for the terminfo compiler.

Besides ncurses, parts of pcurses still survive in 2010, in recognizable form in Solaris.

Who wrote NCURSES?

- Zeyd Ben-Halim started it from a previous package `pcurses`, written by Pavel Curtis. The first widely-used version (1.8.1 in November 1993) was not much larger than pcurses. It added freely-available examples such as `worm`, and a crude configure mechanism.
- Eric Raymond continued development, apparently starting in late 1993 or early 1994.
- Juergen Pfeifer wrote most of the form and menu libraries. Eric added those libraries to ncurses in August 1995, after I had started working with Eric and Zeyd in March 1995, before ncurses 1.9 was released.
- I've done most of the configuration work, do most of the ongoing development and testing.
- numerous people (more than I know about) contributed fixes.

  For details, see the NEWS file in the source distribution. It accurately reflects contributions since 1.9.9e.

In preparing copyright transfer in 1997, I identified more than 20 contributors based on my software archives.

These individuals were cited in the agreement as having contributed more than 20 lines of code each:

```
Heinz-Ado Arnolds, Jeremy Buhler, Andrey Chernov, J.T. Conklin,
Ulrich Drepper, Juergen Ehling, Werner Fleck, Per Foreby, Gerhard
Fuernkranz, Anatoly Ivasyuk, Andrew Kuchling, H.J. Lu, Alexander
V. Lukyanov, David MacKenzie, Rick Marshall, Hellmuth Michaelis,
Tim Mooney, Philippe De Muyter, Eric Newton, Andreas Schwab,
Jesse Thilo, Warren Tucker, Peter Wemm.
```

In addition to Florian La Roche, who agreed to act as maintainer, these were the principal authors of ncurses, for assigning copyright to the Free Software Foundation:

- Zeyd M. Ben-Halim
- Thomas E. Dickey
- Juergen Pfeifer
- Eric S. Raymond

Pavel Curtis' work is in the public domain, hence not needed for copyright assignment. (The `README` file in the ncurses distribution also identifies the authors).

Florian acted as maintainer for about a year. I continued to do the bulk of development, and prepared the 5.0 release. After Florian left unexpectedly (before 5.0), I resumed my pre-4.2 role as the project maintainer.

How can it be distributed?

The major ncurses developers (exclusive of Pavel Curtis, who put his work in the public domain several years before) early in 1998 assigned their copyright to the Free Software Foundation, which promised to use the following distribution terms for at least five years.

```
Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in the
Software without restriction, including without limitation the rights to use,
copy, modify, merge, publish, distribute, distribute with modifications,
sublicense, and/or sell copies of the Software, and to permit persons to whom the
```

```
Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE ABOVE
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name(s) of the above copyright holders
shall not be used in advertising or otherwise to promote the sale, use or other
dealings in this Software without prior written authorization.
```

Is it Free Software or Open Source?

Ncurses is free software. It is not `open source'.

That term applies to a mixture of proprietary software and quasi-free software, and is being promoted currently by several people for a variety of reasons: some as a compromise (in the pejorative sense) between free software and proprietary, and others to take credit for brokering the release of some proprietary software under less stringent conditions.

By relabeling free software (and revising the order of causes and events), the supporters of `open software' are doing the development community a disservice.

Is it GPL'd?

Surprisingly, some people cite ncurses as an example of GPL or LGPL. The copyright notice (which is the above-quoted license) appears 577 places in the 5.1 sources, including all of the header files. Presumably therefore, these people have not actually looked at ncurses.

Adding to the confusion are misleading comments such as this:

```
In particular, if you intend to port a proprietary (non-GPL'd) application using
Cygwin, you will need the proprietary-use license for the Cygwin library. This is
available for purchase; please contact sales@cygnus.com for more information. All
other questions should be sent to the project mailing list
cygwin@sources.redhat.com.
```

By omitting some of the facts, this paragraph states in terms of ncurses, for example, that I cannot work on ncurses on cygwin without buying a license for Cygwin. The same applies to the developers of about half of the contributed software for Cygwin, since not all are GPL'd. There is a better attempt at explaining Cygwin licensing here, but the other page does not use it:

```
This means that you can port an Open Source(tm) application to cygwin, and
distribute that executable as if it didn't include a copy of
libcygwin.a/cygwin1.dll linked into it. Note that this does not apply to the
cygwin DLL itself. If you distribute a (possibly modified) version of the DLL you
must adhere to the terms of the GPL, i.e. you must provide sources for the cygwin
DLL.
```

Probably more people read the FAQ (and are misled) than read the licensing page.

This type of license, by the way, is often referred to as "MIT-style", referring to the MIT X distribution terms. Before assigning copyright to the FSF, substantial portions of ncurses were copyrighted in this style. The main restriction that affects most people is that the copyright notice must be kept on copies—or portions of the copies. That is not done in this online reference, which documents an older version of ncurses. The translation from manpage to html retains the content, but removes the copyright notice, which one may observe is not permitted. Compare with this (copyright notices are retained in the online content, as you can see in the source-view of the page).

For what it's worth, the agreement which we (original ncurses developers) made with the Free Software Foundation reads in part:

```
The Foundation promises that all distribution of the Package, or of any work
"based on the Package", that takes place under the control of the Foundation or
its agents or assignees, shall be on terms that explicitly and perpetually permit
anyone possessing a copy of the work to which the terms apply, and possessing
accurate notice of these terms, to redistribute copies of the work to anyone on
the same terms. These terms shall not restrict which members of the public copies
may be distributed to. These terms shall not require a member of the public to pay
any royalty to the Foundation or to anyone else for any permitted use of the work
they apply to, or to communicate with the Foundation or its agents in any way
either when redistribution is performed or on any other occasion.
```

As is well known, that precludes relicensing to the GPL in any version, since it would place restrictions on which programs may link to the libraries. That would deprive a substantial fraction of the current user base of the use of subsequent versions of the software. No such restriction exists in the ncurses license.

Will it ever be GPL?

I have never considered it a possibility (see the preceding section). It would make the package unusable for most of its current user base, because GPL is a more-restrictive license than MIT-X11 or any of the similar BSD licenses.

However, since the FSF is the copyright holder, it is not impossible that ncurses might someday be relicensed under the GPL. In that case, I would continue development based on the previous version, using the existing license—the one to which I initially agreed.

The original agreement stated that changes which I made to the source would be copyright by the Free Software Foundation. That clause expired after five years (in 2003). It does require written notice (for instance today is June 25, 2007), so in the event of serious disagreement with the FSF, this webpage satisfies that. It is worth noting that all changes that I have made since the most recent release would be in that event copyright by me.

What about the tack program?

That is not part of ncurses. As a convenience (to reuse library functions that are part of `tic` and `infocmp`), it has been distributed with ncurses since before 5.0 (patch date 990414).

However, `tack` is licensed differently: the GNU General Public License (GPL).

This confuses some packagers, who then label ncurses as *GPL*. Most packagers correct the designation when requested. Some do not. To avoid this confusion, it was removed from the ncurses distribution in 2007, shortly after 5.6 release.

What about the terminfo database?

Copyright and Licensing

The terminfo database is a special case. Ncurses provides a different version of the `terminfo.src` file originally collected by Eric Raymond. The ncurses file is not maintained by Eric Raymond, since the agreement which transferred control to FSF states:

```
We hereby agree that if we have or acquire, or any one of us has or acquires,
hereafter any patent or interface copyright or other intellectual property
interest dominating the Program (or use of the same), such dominating interest
will not be used to undermine the effect of this assignment,
```

Changes made to this file are (unsurprisingly) copyrighted via the Berne convention. No explicit "Copyright ©" is required. It only requires that the author be identified (and this is done in the history comments at the end of the file):

```
(1) In order that the author of a literary or artistic work protected by this
Convention shall, in the absence of proof to the contrary, be regarded as such,
and consequently be entitled to institute infringement proceedings in the
countries of the Union, it shall be sufficient for his name to appear on the work
in the usual manner. This paragraph shall be applicable even if this name is a
pseudonym, where the pseudonym adopted by the author leaves no doubt as to his
identity.
```

and noting the first comment in the file states that it is maintained as part of ncurses, this applies:

```
(3) In the case of anonymous and pseudonymous works, other than those referred to
in paragraph (1) above, the publisher whose name appears on the work shall, in the
absence of proof to the contrary, be deemed to represent the author, and in this
capacity he shall be entitled to protect and enforce the author's rights. The
provisions of this paragraph shall cease to apply when the author reveals his
identity and establishes his claim to authorship of the work.
```

Attribution

Occasionally someone in a newsgroup posts a terminfo which has been exported using `infocmp`, saying that it is theirs (and even written by them). Sometimes the claim is true, though more often the data is identical to that from ncurses or a package which includes ncurses. The latter case is interesting.

For instance OpenQM uses terminfo entries which are obtained from ncurses using `infocmp`. In discussion, one aspect glossed over was that some of the content was copied not from ncurses itself but from a *packager's* patch which merged a xterm terminfo which I wrote. Ultimately, the responses from that discussion boiled down to saying that they found it and it's theirs. (The maintainer did agree to add a comment noting the origin of the "public domain" entries).

Other Versions

Eric Raymond's website has an old version misleadingly numbered "11.0". It actually is much older than ncurses's terminfo (whose major version I have left as "10").

The content of "11.0" is derived from ncurses 5.0. It makes more than one change, but most are cosmetic (e.g., reordering the entries within the file, adding about 300 lines of comments—in an 18,655 line file—to make the reordering look nicer).

None of the added comments are useful.

It also modifies the changelog from ncurses to make it appear that people who reported problems to me were the ones who did the subsequent investigation and patches. He had done the same thing prior to ncurses 4.2, for several months in 1997 and 1998, copying changes from ncurses development, and then later "revising" the change history. (I later restored that portion of the changelog).

The last version (11.0.1) copies one of my fixes from ncurses, (prompted by a bug-report by Tijs Michels on the rxvt-workers mailing list in January 2000, in response to ESR's announcement of "11.0.0"). In the mailing list, I pointed out the reason for my change. My original change comment in 1997 said

```
# 10.1.16 (Sat Dec 13 19:41:59 EST 1997)
#      * remove hpa/vpa from rxvt, which implements them incorrectly.
#      * add sgr0 for rxvt.
#      * remove bogus smacs/rmacs from EMX descriptions.
```

GNU termcap 1.3.1 distributes "11.0.1", generated using ncurses' tic program. Disregarding the issue that the file is old and faulty, the reason for generating the file is that ESR's website does not provide a version which resolves all but the last "`tc=`" inclusions.

How big is it?

The answer depends on why you are asking. Some of the reasons for asking include

- does it have some given feature.
- will it fit in some limited-size embedded application,

Growth of the feature set

As noted, ncurses began as a clone of SVr4 curses (up til around 1995). Later, The Open Group began a specification of curses (XSI Curses), which is now known as X/Open Curses. That provided for "wide-characters", such as Unicode, although at that point Unicode was not the obvious answer.

There were a few functions (such as `attr_get`) which were added into the ncurses library in anticipation of XSI Curses. However, I chose to implement the wide-character support using a different library name, "ncursesw". Doing that allowed me to maintain compatibility with applications that used the existing "ncurses" library.

Given that, the "ncurses" library would be comparable to a SVr4 implementation (such as Solaris, IRIX64 or HPUX), while the "ncursesw" library would be comparable to one of the XPG4 implementations (Tru64 being the notable implementation). The ncurses (and ncursesw) library provide some extended functions not found in SVr4/XPG4. A few implementations (PDCurses and NetBSD curses) provide some of these extensions.

Later, I added functions to support simple threaded applications, and Juergen Pfeifer extended that. Again, to maintain compatibility, this is normally built as a a new library name, "ncursest" or "ncursestw". At this time (mid-2011), there is no other implementation of these features.

The ncurses and ncursesw libraries are reasonably source-compatible. That is, an application written for "ncurses" will build with "ncursesw". But it will behave differently in response to your locale settings. (Some distributors, who do not care about the differences, have chosen to merge the names together as "ncurses").

A few applications require changes to use "ncursest", since internal details of the `WINDOW` object are not directly visible in the latter. However, the "ncurses" library has macros and functions which address this area.

Every implementation of curses uses both macros and functions to provide their features. ncurses follows the XPG4 convention where all macros (except for those such as `getyx` which *must* be implemented solely as macros) are also implemented as functions.

Here are counts comparing ncurses 5.9 with other implementations:

Implementation	Macros	Public symbols
ncursestw (sp-funcs)	219	541
ncursest (sp-funcs)	153	439
ncursestw	219	429
ncursest	153	332
ncursesw	219	454
ncurses	153	357
Tru64 5.1	228	481
Solaris 10 XPG4	180	374
Solaris 10 SVr4	199	422
IRIX64 6.5	195	394

HPUX 11.23 XPG4	162	472
HPUX 10.20 XPG4	172	468
HPUX 10.20 SVr3	197	323
AIX 7.1	207	480
PDCurses 3.4	9	364
NetBSD 5.1	86	367

In each case, internal functions are not counted.

The Unix implementations include undocumented features for compatibility with older curses implementations that are not provided by ncurses.

Both PDCurses and NetBSD curses contain functions not counted here because they are not relevant to a comparison with SVr4/XPG4 curses. For example, PDCurses in X11 contains functions for initializing the X window. On the other hand, relevant extensions such as PDCurses' version of `wresize` are counted.

Types of library users

PDCurses on the other hand does not include the functions used to obtain terminfo information. That does not prevent it from being a curses implementation. X/Open Curses' documentation treats those separately, allowing for the possibility of curses implementations without terminfo (or termcap either for that matter).

That raises another issue: what types of interfaces do curses libraries (and ncurses in particular) support?

ncurses includes the conventional curses interfaces, with extensions (new functions) in each interface. The interfaces are:

- form, menu: upper-level libraries provided in a few systems such as SCO and Solaris.
- panel: an upper-level library which should have been standarized as part of the curses library.
  Again (outside of ncurses), it is provided by SCO and Solaris.
- curses: the higher-level interface used by *all* curses applications.
- terminfo: a lower-level interface used by some applications.
- termcap: a lower-level interface used by some applications.

Curiously, NetBSD provides variants of the form and menu libraries, but lacks the panel library. Conversely, PDCurses provides a panel library but lacks the form and menu libraries. Because NetBSD uses *opaque* structures, it is difficult to write portable applications using their form and menu libraries. One might have better luck with PDCurses' panel library, which has the same ancestor as ncurses' panel library.

The lower-level interfaces rely upon the application (rather than the library) to decide how to put characters on the display. That is referred to as "optimization", e.g., in the documentation:

```
The ncurses library routines give the user a terminal-independent method of
updating character screens with reasonable optimization.
```

Most applications made the transition to using the curses interface long ago. There are some applications which use one of the lower-level interfaces. In some cases there are technical reasons for this. Inertia is more often the reason.

Here are some examples of well-known applications which use curses:

- lynx
  A curses application from the beginning, lynx can be built with any version of curses, including PDCurses and VMS curses.
- nvi
  Keith Bostic introduced nvi in 4.4BSD, revised it for 4.4BSDLite in 1994. At that point, it was using curses (preferably System5 curses). Lacking that, it had a bundled copy of 4.4BSD curses, which sufficed. Bostic's interest in ncurses stemmed from his desire to eliminate the bundled 4.4BSD curses, and he was helpful in many ways, both technical (with bug reports) and otherwise.
- tin
  Originally in K&R C using termcap, I modified it to use curses. Most of the subsequent work to handle wide-characters was done by others.

Widely-used *termcap* applications include these:

- screen
- w3m
- vim

Applications which use only terminfo are less well-known. For example, there is tack. Because the terminfo and termcap programming interfaces are similar, it is trivial to make an application build with terminfo if it was originally written to use termcap. A few (such as vile) do exactly that.

Going the other way is not necessarily simple, because the terminfo interface provides a data structure `TERMTYPE` holding all of the terminal's information which eliminates the need for calls to the tigetstr, tigetnum and tigetflag functions. If an application (such as *tack*) is written to use that, it is difficult to make it use termcap. Additionally, ncurses provides extended capabilities in its terminal database which are not necessarily visible to termcap applications. Xterm, in its `tcapFunctionKeys` feature uses these capabilities; gnome-terminal in its analogous termcap interface cannot.

Because of these differences, it is best to speak of termcap-applications, terminfo-applications and curses-applications. Curses-applications may use terminfo functions, but the reverse is not true. Given the terminfo interface, there is no reason for a curses-application to use the termcap interface.

An easy way to distinguish between these types of applications is to see what library calls they make:

- curses: initscr or newterm
- terminfo: setupterm
- termcap: tgetent

Even this categorization is simplified:

- Some curses applications (such as *nvi*) switch between full-screen and normal modes of operation, e.g., to support the "open mode" of *vi*. To accomplish that, *nvi* uses lower-level calls at times, e.g., to erase characters.
- *nvi* (and dialog) treat xterm's alternate screen feature specially, using low-level workarounds to disable the feature as they switch to/from full-screen mode.
- *nvi* (and vile, using its curses driver) do not use the curses library's input function wgetch. That is because *vi* treats the *escape* character specially. Consequently, the curses decoding of function- and special-keys is not usable by *vi*-like programs.

Nothwithstanding their use of low-level calls, these are curses applications because they use the library to update the display. Applications which do their own display optimization (such as *vim*) are not curses applications.

Relationship with vi

Not all implementations of *vi* are curses applications. In fact, considering it carefully, most are not.

One likely objection to that statement is the generally accepted notion that Ken Arnold "took" functions from Bill Joy's *vi* to make his library. For instance, in the preface to *UNIX Curses Explained*, Goodheart said (in 1991):

```
It all started in the late 1970s when Bill Joy, in writing his editor ex (probably
more famous by the name vi nowadays), wrote a set of routines which read a
terminal capability database. The database, then named termcap, generally
described how to manipulate individual terminals and what they where capable of.
The routines he created, which accessed the termcap database, implemented optimal
cursor movement.

Kenneth C.R.C. Arnold took these routines almost without changes and derived from
them what is known today as the curses package.
```

People tend to refer to the second paragraph without noticing the first one. The *"where"* versus *"were"* typographical error in the first paragraph is an error by the publisher. The last sentence of the first paragraph is an error by the author: termcap does not, never did do that.

Optimal cursor movement (cursor optimization) in *ex* was done in the remainder of the application. The termcap functions were readily reusable; the remainder was not. Arnold added to Joy's original idea by making the library *reusable*.

The Unix Heritage Society has copies of 3BSD, 4.2BSD, 4.3BSD (multiple). Using those, one can make useful comparisons :

- ex_vis.h file from *ex* in the 4.2BSD distribution
- curses.h from curses library in the same distribution.

Joy's header file is littered with separate variables. Arnold's header file organizes those into structures, notably `WINDOW`. The `vtube` array in the former corresponds to the bulk of the storage used by `WINDOW`. In the same way, Arnold recognized useful aspects of terminal manipulation scattered through *ex*, and organized those into a library which improved on the original design. Aside from `beep`, there is no straightforward mapping between the functions in the two programs.

Traditional *vi*, however, does not use curses. With the release of 4.2BSD, it lost its private copies of the termcap functions, and, in the main branch of development, at AT&T, became a terminfo application. With 4.3BSD, the termcap-based *vi* (and derived code such as "heirloom vi") was no longer the main line of development. Bostic's *nvi* was a re-implementation (based on *elvis*), done specifically to disentangle the BSD project from AT&T source code.

Switching to terminfo was simple. Switching back would be hard, since the AT&T *vi* uses the `TERMTYPE` structure for accessing the terminal capabilities. AIX, HPUX, IRIX64, Solaris all use the AT&T-derived *vi*.

Growth of the library filesize

These are some notes for further development:

- shared vs static libraries
- libtic / libncurses / libtinfo
- slang vs libtinfo

- slang vs libtermcap
- slang overlapping windows

What platforms does it run on?

It should build and work on any POSIX platform. It also works on some platforms that are non-POSIX.

Current development is focused on the wide-curses configuration (ncursesw). These platforms are known to work:

- BSD variants:
  - FreeBSD 9.0, 8.2, 7.0, 6.4, 4.9
  - OpenBSD 4.9
  - NetBSD 5.1
- Linux-based systems
  - Slackware 13.x
  - Debian 6.0, 5.0, 3.1
  - Fedora 13-16, CentOS 4-6
  - Mandrake 2010
  - SuSE 11, 12

The normal (8-bit character) configuration is known to work on the same platforms. I've also built these in preparation for the current release):

- AIX 5.3, 5.1 (cc, gcc)
- BeOS R5
- Cygwin
- HPUX 11.23, 11.11, 11.00, 10.20
- IRIX64
- Mac OS X 10.7
- OS/2 EMX 0.9d (gcc 2.7.2)
- QNX 6.1
- SCO OpenServer 5.0.5e (cc/CC, gcc 2.7.2.3)
- Solaris 10, 9, 8, 7, 6, 2.51
- Tru64 (aka OSF1 and Digital Unix) 4.0d, 5.1 (cc)
- Windows 7, using MinGW.

What is the latest version?

The current version is 5.9 20110404, available at the GNU ftp site.
Ftp: /pub/gnu/ncurses directory

Ftp: ncurses.tar.gz (gzip'd tar)

I also maintain patches toward the next release (5.10 or 6.0). See the NEWS file for changes.

Ftp: ftp://invisible-island.net/ncurses/5.9/

Official releases

There have been a number of releases of ncurses. Some are available on CDROM (beginning with 1.9.4), and are archived on various ftp servers. If you are downloading, however, the older versions are of limited interest except for software testers.

- 5.9 (4 April 2011). This fixes a regression in `newwin`, and improves configurability/portability of the Ada95 binding when built as a separate tree. (26 February 2011). This extends support for threaded applications by providing a new API which eliminates the need for a global screen-pointer. It also introduces a port to Windows using MinGW, as noted in the 5.8 announcement.
- 5.7 (2 November 2008). This supports support for threaded applications. It also distributes `tack` separately.
- 5.6 (17 December 2006). This supports hashed database for the terminal descriptions. It also improves support for magic cookies.
- 5.5 (10 October 2005). This improves support for wide- and multibyte characters, modifies the form- and menu-libraries to work with multibyte characters. It also improves support for cross-compiling.
- 5.4 (8 February 2004). This improves support for wide- and multibyte characters, implements the remaining parts of the X/Open Curses interface. It also improves support for termcap.
- 5.3 (12 October 2002). This provides support for wide- and multibyte characters, implements most of the corresponding X/Open Curses interface. It also improves support for termcap.
- 5.2 (21 October 2000). This provides better support for termcap, fixes for the manpage and shared library configurations, and improved checks for corrupt terminfo database.
- 5.1 (8 July 2000). This provides better support for termcap as well as implementing new extensions for colors.
- 5.0 (23 October 1999). This aligns interfaces to match version 2 of the X/Open Curses Single Unix specification from 1997. It also provides better termcap support, and termcap-like features such as user-defined terminal capabilities.
- 4.2 (2 March 1998). This implements C++ bindings for the library, as well as extensions to support runtime definition of

keycodes.
- 4.1 (16 May 1997). This release features support for GPM (mouse on Linux console), and adds an extension for default colors.
- 4.0 (24 December 1996). A change in version numbering addresses a problem exposed by David Engel's `ld.so.1.8.5` program for Linux.
- 1.9.9g (1 December 1996).
- 1.9.9e is broken. A last-minute/untested change causes forms and menus to not refresh. Foreground/background colors are combined incorrectly, working properly only on a black background.
- 1.9.8a (ok)
- 1.9.7a (ok)
- 1.9.4 is the oldest version that you should consider installing. It comes with Slackware 3.0; earlier releases have a number of problems, including incompatible terminal descriptions.

What other programs are there?

The ncurses distribution only includes programs that must be maintained with it, since they rely on internal details of the library. Here are pointers to more generic curses programs, which can be built with ncurses or similar implementations:

- cdk Curses Development Kit (a library of widgets)
- dialog Script-driven curses widgets

Known/Frequent problems

Building and Installing NCURSES

How do I run the test-programs?

You must first install the terminfo data (i.e., "make install.data").

On many systems (those that have a SVr4 curses installed) you can run the test programs using the vendor's terminfo database (e.g., Solaris, IRIX, HP-UX) by setting the TERMINFO variable to point to that instead.

How do I apply patches?

See also How are patches organized?

I name development patches after the base release, with the patch date originally (in 1996) in the form *yymmdd* and starting with the year 2000 *yyyymmdd*.

- you need patch 2.1 (originally, I believe, in the X distribution, but also from GNU now, e.g., ftp.gnu.org). Version 2.5 offers some advantages (handles lines wider than 1024 characters), but introduces new bugs (its name resolution algorithm gets more easily confused by duplicate names in the tree). The interim versions (2.2, 2.3, 2.4) are unacceptable.
- you also need gzip 1.2.4, since both the base release and the patches are gzip'd.
- Assuming you have all of the patches and the 4.2 tar.gz file in the same directory and are running in Bourne shell:

```
zcat ncurses-4.2.tar.gz |tar xf -
cd ncurses-4.2
for n in ../ncurses-4.2-*.gz ; do zcat $n | patch -p1 ; done
```

Where are the patches?

Development patches (and a rollup patch updated periodically) are available for download: patches to 5.9 Look for a file named something like "`patch-5.9-`*`yyyymmdd`*`.sh.gz`" The "*yyyymmdd*" part corresponds to the patch-date.

After providing a rollup patch, I remove the development patches to reduce clutter. Beginning in May 2013, I modified the process to provide all of the development patches since the release in "dev-patches.zip".

There are also a few rollup patches between releases:

Download:

- patches
- patch-4.2-5.0.sh
- patch-5.0-5.1.sh
- patch-5.1-5.2.sh
- patch-5.2-5.3.sh.gz
- patch-5.3-5.4.sh.gz
- patch-5.4-5.5.sh
- patch-5.5-5.6.sh.gz
- patch-5.6-5.7.sh.gz
- patch-5.7-5.8.sh.gz
- patch-5.8-5.9.sh.gz

The rollup patches include all patches through the cited version. You must apply them to the base release, e.g.,

```
zcat ncurses-4.1.tar.gz |tar xf -
cd ncurses-4.1
sh ../patch-4.1-4.2.sh
```

I have tested the rollup patches with patch 2.1 and 2.5, adjusting for their respective limitations.

The terminfo database is big—do I need all of that?

Not at all. You can load a subset of the terminfo database. I use a variant of this script to load the terminal descriptions that I need on my machine:

```
#!/bin/sh
# uses the -e switch on tic to selectively load descriptions that are
# interesting for my testing.
if test -f terminfo.src ; then
        TMP=/tmp/load$$
        trap "rm -f $TMP" 0 1 2 5 15
        tic -s -1 -I -e'
ansi, ansi-m, color_xterm, ecma+color,
klone+acs, klone+color, klone+sgr, klone+sgr-dumb,
linux, pcansi-m, rxvt, vt52,
vt100, vt102, vt220, xterm' terminfo.src >$TMP
        tic -s $TMP
else
        echo 'oops'
        exit 1
fi
```

Do I really need a terminfo database?

You could compile-in fallback definitions for the most commonly used terminal types. To do this, you must already have `infocmp` installed (note that ncurses 5.0 infocmp's support for fallback descriptions is done differently from ncurses 4.2).

But fallback definitions are really only useful in embedded applications, where no external files are wanted.

Which terminfo database do I need?

The most reliable terminfo database is that distributed with ncurses 5.0, or via followup development patches. The original process of incorporating terminal descriptions from various sources corrects some errors in the originals, but introduces others (either translation errors, or misconceptions). Besides working to resolve these, from time to time we incorporate new sources.

As noted in 29 February 2004, the terminfo database at http://www.catb.org/~esr/terminfo/ does not appear to be actively maintained. Since the release of ncurses 5.0 in late 1999, there are numerous fixes which are not in that database.

As of January 2012, this is still true. A check with

```
infocmp -x -F termtypes.master terminfo.src
```

reports that there are 1807 differences for the entries which the two have in common, as well as 325 entries not in the "master" terminfo database.

Here are links to current versions:

- terminfo.src.gz
- termcap.src.gz

If you choose to not install the ncurses terminfo database, we have found that the SVr4 systems (Solaris, IRIX 6, OpenServer and HP-UX 10) work well enough for many purposes. Other systems either are not binary compatible, are incomplete, or contain more errors than either of the choices mentioned. Some that are not binary compatible can be accommodated by configuring ncurses to use the native terminfo format. These include AIX, HP-UX 11, Tru64 and U/Win.

Is ncurses terminfo compatible with my system?

Sometimes.

Terminfo is compiled into binary form, with booleans, numbers and strings in arrays. As long as the array items line up, and the headers (that tell how long the arrays are) are compatible, ncurses and your vendor's system can each use the same terminfo database. Older sytems (e.g,. those based on SVr3) implement a subset of the SVr4 terminfo. For example, HP-UX 9 is "compatible" up to the entries that would describe graphic (box) characters. There it diverges.

Other systems (e.g., Digital Unix 4.x and the older AIX 3.2.5) use different formats and are not compatible.

However, terminfo *source* is compatible and can be compiled using the appropriate tool (usually *tic*). Some terminfo descriptions may produce warnings (e.g., the memu/meml capabilities in the standard xterm distribution), but the tools compile what they recognize and warn about the rest.

The ncurses *tic* program recognizes a wider range of input than other terminfo compilers, including extensions coordinated with *infocmp* that make it easier to use:

- compile descriptions that use the long names that "infocmp -L" produces.
- the "infocmp -f" option formats complex expressions into indented format. These are usable with the standard SVr4 tic program, though not with ncurses 4.0 and below due to a bug.

Is the ncurses library compatible with my system?

Yes/no.
Source compatible yes, binary compatible no.

You cannot simply replace your existing curses or termcap library with ncurses unless you are prepared to recompile applications that use the curses or termcap library (e.g., `vi`, `telnet`).

For systems using the Linux kernel, that is feasible. But not Solaris or other proprietary systems. For those, I recommend configuring with the `--disable-overwrite` option. This directs the configure script to install the library so you would link it as `-lncurses`, not adding a symbolic link to make it link as `-lcurses`.

The `--disable-overwrite` option also installs the header files such as `curses.h` in a subdirectory, e.g., `/usr/local/include/ncurses/curses.h`, thereby avoiding a (mis)feature of recent versions of `gcc` which look first in `/usr/local/include` for header files. Since the vendor's compilers do not do this, a common problem results: compiling with `/usr/include/curses.h` and linking with `/usr/local/lib/libcurses.a`.

Starting with ncurses 5.3, the behavior for this option changed. If you do not install into `/usr`, the configure script will assume you do not wish to overwrite the existing version of curses. Configure scripts which do not check for ncurses headers in both locations are incorrect anyway. They can be accommodated by setting the `$CPPFLAGS` environment variable, e.g.,

```
setenv CPPFLAGS "-I/usr/local/include/ncurses"
```

Problems with Cross-Compiling

I have done occasional cross-compiles of ncurses since 2003, using DJGPP as a target, and recently (writing this in 2010) some MinGW builds. For example

```
TARGET=mingw32
configure \
        --with-build-cc=gcc \
        --host=$TARGET \
        --target=$TARGET
make
```

Cross-compiles of ncurses require compiling utilities which are then executed at build time. The utilities are wrappers around ncurses library functions, which means that some special #define's may be needed to compile them. In particular (until May 2010) cross-compiling the wide-character library required adding something like this:

```
--with-build-cppflags=-D_XOPEN_SOURCE
```

Ncurses 5.7 introduced a separate problem. The terminal database distributed with it uses a feature that causes older versions of tic to hang. If you are cross-compiling, the database is installed using your system's copy of tic. You can work around the problem by either updating ncurses (5.7 was released in 2008), or substituting an older version of `misc/terminfo.src` in the build tree.

Configuring NCURSES

Do I need all of those programs and libraries?

Well, I use them...

Wrong answer.
You may need only the ncurses library, or even just the terminfo database. The top-level Makefile in the build tree is designed to let you install various combinations according to your requirements. But there are a few constraints:

- You can install only the terminfo database by typing "make install.data". But you must have a copy of `tic` built. If you chose to build with shared libraries, you should first install the libraries and programs, e.g., with "make install.libs install.progs". Otherwise the shared libraries will not be found properly for the terminfo compiler to run.
- The version numbers attached to the shared libraries do mean something. Do not rename them (or link other versions to them).
  Version 5 is not compatible with version 4.
  Version 4 is not compatible with version 3.
  We have made corrections at each release to match the X/Open Curses interface specification. I have seen as many people complaining about mysterious problems with ncurses as I have seen people advising others to save time/space by linking them.

Do I need the C++ binding?

You may not need the C++ binding for ncurses. However, you should configure ncurses with C++ if it is available on your system. Otherwise, you will not be able to compile ncurses applications with the C++ compiler.

This point is not clear in the INSTALLATION instructions, having been thought too obvious to dwell upon. However, some distributors have "customized" ncurses, omitting the C++ binding to save space (or the time to issue separate "make install" commands for the components which they really need).

The problem is this:

- Both ncurses and C++ declare the *bool* type. It is part of each of their respective standards.
- There is no standard that says what the size of *bool* is.
- The ncurses interface uses *bool*.

The ncurses configure script determines the actual size used for *bool* by the C++ compiler on your system. If you suppress this configuration check, the default size for *bool* is not guaranteed to work with your compiler.

With 5.0, the configure script provides two options (`--without-cxx` and `--without-cxx-binding`). Use the former to suppress the configure checks for the C++ compiler, e.g., when there is no working C++ compiler on your system. Use the latter to omit the C++ binding, if you must.

Why does *make menuconfig* fail?

I can only guess (people having trouble in this area generally do not answer email ;-)

The Linux "make menuconfig" attempts to build a customized dialog program called `lxdialog`. This uses ncurses, which of course is why you are reading this question/answer.

- The development libraries or header files for ncurses may not be installed.
- The libraries may be installed in an unusual place. But in any case, */usr/src/linux/scripts/Makefile* is simple to read, to see where `make` looks.
- You may have attempted to upgrade your C library, and did not complete the job by upgrading libraries that depend upon it.

Problem building with gcc 2.96.x

The C preprocessor in gcc 2.96 is broken. In particular, the C preprocessor has multiple bugs including

- does not handle whitespace correctly (breaks ncurses).
- cannot compile its own preprocessor output (breaks lynx).

These were apparently fixed in gcc 3.0.4 (do not waste time with gcc 3.0).

Problem building C++ interface with gcc 3.x

Since releasing ncurses 5.2, this has been the most frequent reason for referring people to the rollup patch. The problem is that the C++ bindings to C functions such as `vsscanf()` were altered (more than once, apparently as an afterthought) in preparing the gcc releases. Since the gcc changelog does not cite these changes except obliquely, and they are undocumented, it is possible that they may change again.

Testing for Memory Leaks

Perhaps you used a tool such as `dmalloc` or `valgrind` to check for memory leaks. It will normally report a lot of memory still in use. That is normal.

The ncurses configure script has an option, `--disable-leaks`, which you can use to continue the analysis. It tells ncurses to free memory if possible. However, most of the in-use memory is "permanent".

Any implementation of curses must not free the memory associated with a screen, since (even after calling `endwin()`), it must be available for use in the next call to `refresh()`. There are also chunks of memory held for performance reasons. That makes it hard to analyze curses applications for memory leaks. To work around this, build a debugging version of the ncurses library which frees those chunks which it can, and provides the `_nc_free_and_exit()` function to free the remainder on exit. The ncurses utility and test programs use this feature, e.g., via the `ExitProgram()` macro.

Customizing NCURSES

How do I set up a private terminfo database?

If you must maintain your own terminfo database, SVr4 curses and ncurses both use the $TERMINFO variable to override the standard location of the terminfo database. Ncurses also provides two related extensions: the $HOME/.terminfo directory and the $TERMINFO_DIRS search path.

Though ncurses tests $TERMINFO first, otherwise it reads from $HOME/.terminfo before the standard location, and writes to that location after failing in other places. This design is a compromise which is made more complicated if you have configured ncurses with the --enable-termcap and --enable-getcap-cache options. Unless you are prepared to deal with the hidden conflicts, you should simply remove the $HOME/.terminfo directory.

If you do not wish to use $HOME/.terminfo (and are not able to replace ncurses on your system), ncurses 4.2 and later work properly if you replace that directory with a file so it cannot write terminfo entries which would conflict with the standard location.

The toe program can show a side-by-side comparison of terminal databases, making it simple to see conflicting entries from your private terminal database.

As noted, ncurses also provides the $TERMINFO_DIRS extension. Unlike $HOME/.terminfo, this allows you to specify one or more locations for the terminal database. When reading a terminal description, ncurses chooses the first one found. Using this feature, you can tell ncurses to look in your own database(s) as well as one or more system-defined locations.

The usual reason for creating a private terminal database is to work around inability to change the system's terminal database. Setting $TERMINFO suffices for most users, and happens to work with other implementations than ncurses. However, for non-ncurses implementations, it limits the user's environment to just the terminal database indicated by the $TERMINFO variable.

As noted in "Which terminfo database do I need?", there are a few older systems whose compiled terminfo files differ slightly from the SVr4 layout used by ncurses. While ncurses can be compiled to match the system's format (I do this), it seems to be a less-used feature. Packagers prefer the simpler approach of letting ncurses use its database and leaving the system applications alone. There are then these cases for constructing private databases in an NFS-mounted home directory, shared across different operating system types:

- the system's terminfo format is SVr4-compatible.
    - Use the $TERMINFO variable, e.g., to have all applications (ncurses and otherwise) use the same terminal descriptions.
    - Use the $TERMINFO_DIRS variable to make ncurses applications see a distinct set of terminal descriptions from the system, e.g., for "xterm".
- the system's terminfo format is not SVr4-compatible (AIX for instance).
    - if ncurses is compiled to match the system's terminfo format
        - as before, you can set either $TERMINFO or $TERMINFO_DIRS according to whether you want to use the same definitions for all applications (ncurses and system).
        - choose a different location for the database, e.g., $HOME/.terminfo-aix to avoid conflict with other platforms.
    - if ncurses is *not* compiled to match the system's terminfo format, then it is likely that ncurses and the system curses/terminfo libraries cannot read the other's compiled terminfo files.
        - You can set $TERMINFO_DIRS to tell ncurses to look at its own terminal database; the system libraries will not look there.
        - Do not set $TERMINFO, since this will cause both to look in the same place.

A good place to start when customizing a shell initialization script for your private terminal databases is of course `uname`, since it is provided on the systems which are relevant to this topic, and its output can tell which system type is used. Alternatively (if your environment happens to have an inconsistent mixture of ncurses packages), the hostname is another place to get useful parameters for the initialization script.

As a rule, these settings should be done in the shell's *login* script rather than the one which is run on each shell *initialization*.

My terminal is not recognized

Usually this happens because you have not installed the `terminfo` database, or it is not in the proper location. If you do not, and (if ncurses happens to be configured to provide termcap support using the "--enable-getcap-cache" configure option) the application is unable to locate the terminfo database, the ncurses library will attempt to recover by reading `/etc/termcap`, translating it into a private terminfo database, i.e., a directory:

```
$HOME/.terminfo
```

This directory can be a nuisance, because the termcap file often does not contain complete or consistent terminal descriptions. Remove it and correct the problem (i.e., install the `terminfo` database). Better yet, do not enable the feature (it has been disabled by default since late 1996).

You (or the person who configured ncurses) may have installed terminfo in the wrong, or an obsolete location:

- On Linux-based systems, as well as BSD variants such as FreeBSD and NetBSD the preferred location for terminfo is `/usr/share/terminfo`, with a symbolic link from `/usr/lib/terminfo` for compatibility with older applications. Prior to 1.9.9g, the `configure` script defaulted to a /usr/local prefix rather than /usr, necessitating an explicit

  ```
  configure --prefix=/usr
  ```

- On other systems, the prefix defaults to /usr/local. You may wish to change this. Some users install ncurses replacing other versions of curses altogether. I don't do this. If you do, read the INSTALL file.
- Some packagers have gone further, configuring ncurses for a given terminfo location (which is not the system default), and then treating ncurses's terminfo database as "optional". You can always recover in this case by setting the `TERMINFO` environment variable to point to the real terminfo database.
- Finally, prior to 1.9.9g, the actual terminfo directory resided in `/usr/lib/terminfo`. It was moved to `/usr/share/terminfo` to conform to file-system standards. The installation script attempts to determine if you have a real directory at the old location, and will not delete that (because some applications, such as `mc` and `screen` install

special terminal descriptions that you would not want to accidentally delete. The best solution in this case is to move the old directory to the new location and install the new terminfo data, merging into the old data.

Why use terminfo instead of extensible termcap?

Several reasons:

- there is no standard for termcap. That is a drawback to developing applications using termcap. Amusingly enough, many of the termcap capabilities in common use now are defined in terms of terminfo, much like the `inch` is defined in terms of a `centimeter`.
- termcap libraries are not as fast, and have a long history of susceptibility to buffer overflows. See the discussion of tctest for example.
- terminfo supports conditional expressions.
- ncurses terminfo is extensible anyway. It has been since 1999 (release 5.0). If you give the `-x` option to `tic`, it compiles additional information which may be in the terminal description, using the syntax to determine types. If `-x` is omitted, `tic` warns about the unrecognized information.

As an example of how this extensibility feature can be used, consider the case of OpenQM, whose developers copied more than 1000 lines of ncurses library code into their application to support some extended terminal capability features. Here are two patches showing how that can be improved by removing the nonstandard features:

- Make the syntax standard
- Remove the unnecessary code

Making NCURSES Fast

Why does my program scroll slowly?

Besides padding (i.e., time delay) information, which may be slowing your application down on a terminal emulator, ncurses provides two versions of scrolling optimization. The newer/improved version was incompletely tested at the time of release of ncurses 4.2, so it was marked *experimental* in the configure script.

The newer scrolling (hashmap) algorithm does not work properly in older versions of ncurses. Starting with ncurses 4.2, however, we recommend enabling this logic when configuring, using the --enable-hashmap option. It is configured by default in ncurses 5.0

Why does VT100 refresh slowly?

Some terminal descriptions contain padding (i.e., time delay) information. Ncurses uses this information to slow down the rate at which characters are sent to the terminal.

However, the vt100 terminal description, which is one of the most widely used (or misused) contains padding information for a real DEC VT100. It is not suitable as a replacement for the xterm terminal description. (Xterm requires no padding).

If you **must** use the vt100 terminal description, you may consider setting the NCURSES_NO_PADDING environment variable which is implemented in current versions of ncurses (since late 1998). That directs ncurses to ignore nonmandatory time delays in the terminal description.

Making Color Work

How do I get color with VT100?

Sorry. Real vt100's do not do color (ANSI or otherwise). Likewise, vt220, vt340 do not support ANSI colors. You may be running a terminal emulator which does and do not like this explanation.

You get "color with VT100" by running a terminal (or emulator) that supports colors, and by setting up the terminal description so that ncurses knows how to perform basic operations (setting the foreground and background colors). See My terminal doesn't recognize color.

Ncurses does not "know" that your terminal does support color. You must tell it. Some terminals (e.g., the higher models of DEC's VTxxx series) provide status information to a host on the capabilities supported by a terminal. Unix hosts do not interpret this information to set your $TERM environment variable. Instead, $TERM is set based on the connection which you make with your computer (e.g., a device listed in the /etc/gettydefs or /etc/gettytab files). You can override this by setting $TERM to a correct value or setting $TERMINFO to a private database.

Ncurses does not by itself know that vt100's do not do color. The standard reference for VT100 is its reference manual. There is a copy of that on vt100.net (look for EK-VT100-TM-003_Jul82). There are of course contrary sources of information about color-vt100's. The earliest one that I have seen is this, which (besides the misinformation in its opening paragraph) has more than one point where it differs from vt100:

- *Terminal Setup*. The example given refers to modes, saying that few are implemented correctly. But the example itself is incorrect, because it omits the "?" character which tells that the sequence is a DEC private mode. For comparison,

see **DECAWM** in xterm's control sequences document.
- *Fonts*
  - vt100 has one "font". Bold is not a font, but one of the video attributes which include blink, reverse, underline.
  - The term "font" is inappropriate; "character set" is correct.
  - The control sequence cited for switching fonts omits the character which identifies the character set to use.
- *Cursor Control*
  - *Force Cursor Position* is the standard **HVP** (Horizontal and Vertical Position).
  - *Save Cursor* is from ANSI.SYS rather than VT100.
  - *Restore Cursor* is from ANSI.SYS rather than VT100.
- *Erase Screen* states that the cursor moves to home. This is a well-known difference between ANSI.SYS and VT100. With VT100, the cursor does not move. Incidentally, ISO 6429 (ECMA-48) does not mention either behavior.
- *Define Key* is not a VT100 feature. It is a feature of ANSI.SYS.
- *Set Display Attributes*.
  - *Bright*. The standard and VT100 documentation refer to "Bold".
  - *Dim* is not a VT100 feature.
  - *Hidden* is a VT220 feature, not found in VT100.

There are several copies of this document (usually missing the copyright notice) available around the Internet.

After Andrey Chernov added *vt100-color* and others to the FreeBSD termcap file in 2002, I discussed this with him, pointing out that while there are terminal emulators which do this, the hardware did not. He resolved the issue by adding a comment:

```
# For color-enabled terminal emulators
```

(See revisions 1.117 and 1.119).

Even with that clarification, there are others who conflate "ANSI", "VT100" and color, for example

- Andys Terminfo Package
- Debian #241717

Ncurses does not have terminal descriptions such as *vt100-color* because those are inevitably an oversimplification. For related discussion, see my comments on "xterm-color".

My terminal doesn't recognize color

Check the terminal description, to see if it is installed properly (e.g., in `/usr/share/terminfo`) by looking at the output of `infocmp`.

It should contain the following capabilities (shown with infocmp -L):

> `orig_pair` or `orig_colors`
> and `max_colors`
> and `max_pairs`

as well as

> `set_foreground` and `set_background`
> or `set_a_foreground` and `set_a_background`
> or `set_color_pair`

The `orig_pair` and `orig_colors` capabilities are not required in ncurses 5.0 (SVr4 curses works properly without them).

The most common complaint is that "I can see colors using `ls`, but not with ncurses applications". This is due to not having installed the `terminfo` database. GNU `ls` (in contrast to FreeBSD `ls`) uses its own data (with hardcoded SGR numbers and ignoring the back color erase (bce) feature supported in terminfo/termcap), which is unrelated to other applications.

My terminal shows some uncolored spaces

Even if your terminal *supports* the back color erase (bce) feature, you may still have problems using it.

We can blame the standard (ISO-6429 aka ECMA-48 aka "ANSI"). It is too simplistic, being *descriptive rather than prescriptive*. What is that, you might ask? It means that the standards writers used existing simplified descriptions by developers for the features rather than decomposing the features into distinct parts—and then making precise descriptions of those for testing compatibility of various implementations. The standard does not go into enough detail to *prescribe* a particular behavior.

Back color erase covers many (usually) related features, because there are many control sequences which might affect color, depending on the implementation. Here is a list—for each item there is probably at least one terminal type which differs from ncurses' assumptions. The list shows terminfo names with the standard control name, if any, in parentheses:

- *sgr0* (`SGR 0`). The standard notes that this is "implementation-defined", and "cancels the effect of any preceding occurrence of SGR".

For practical purposes, ncurses has to assume that the cancellation applies to the normal video attributes such as bold, underline, reverse, but possibly not affecting color. Color came into the picture fairly late in the history of terminals, and there have been some (such as color_xterm) where `SGR 0` had no effect on color.

- *ed* (`ED`). The erase-display feature is assumed to fill the cleared display (or part of the display) with the current background color.

  Examples of terminals which do not do this include dtterm and teraterm 2.3.

- *el*, *el1* (`EL`). The erase-line feature is assumed to fill the cleared line (or part of line) with the current background color.

  There is no standard requiring that the behavior of `EL` and `ED` should affect color in the same way. It is a design choice; ncurses assumes this.

- *ech* (`ECH`). Erasing a character (which does not move any text on the screen) is assumed to fill the erased cells with the background color.

  Rxvt does not do this, though its behavior matches most of the other assumptions.

  For xterm, `ECH` is a VT220 feature. VT220s (and for that matter, none of DEC's VTxxx series) did not implement ANSI color (or any style of color which you would use with ncurses). Rxvt's behavior was based on its developer's interpretation of DEC's documentation. XTerm's behavior on the other hand was based on (generally) agreeing with the developer of the Linux console.

- *ich*, *ich1* (`ICH`). Inserting an *empty* cell at the current position is assumed to fill the empty cell with the background color.

  Note that this control does not *erase* as one would infer from *back color* ***erase***. But it creates an *empty* cell.

- *dch*, *dch1* (`DCH`). Deleting the cell at the current position shifts the remainder of the line left, introducing an *empty* cell on the right end of the line.

  Note that this control does not *erase* as one would infer from *back color* ***erase***. But it creates an *empty* cell.

- *ind*, *indn* (nonstandard). Scrolling the text up is assumed to fill the newly empty line (at the bottom of the display) with the current background color.

  Indexing, aka "scrolling" is not covered in the standard. The behavior is completely implementation-dependent. This is a commonly used feature of XTerm.

  There are subtle differences in behavior still possible, beyond filling and not filling. For instance, if the scrolling was in response to printing text which wrapped, that may or may not (depending on implementation) fill the newly empty line with the current background color.

  For instance, a change in this area the Linux 2.6.26 console driver in April 2008 was reverted 6 months later. That was a bug for Linux console, because it broke 15 years of predictable behavior. Likewise it would be a bug for XTerm (and I have rejected patches to "improve" its behavior in this detail). But other terminals could do this intentionally as I observed of the "VT340" emulator bundled with the OnNet TCP/IP product of FTP Software in the mid 1990s.

  The standard, by the way, takes it for granted that text will wrap and force the display to scroll. But it uses the terms "wrap" and "scroll" in only a few obscure comments, and doubtless left much leeway for implementation-dependent behavior.

- *ri*, *rin* (nonstandard). Scrolling the display down is assumed to fill the newly empty line (at the top of the display) with the background color.

  Again, there is no standard behavior. There are only design choices and assumptions.

- *csr* (nonstandard). Setting scrolling margins is assumed to limit erasures (and filling with background color) to those margins.

  This is a commonly used feature of XTerm. Vertical scrolling margins are not the only feature of this sort. There are also horizontal margins (VT320) and origin mode (VT100) which can affect the ability to move the cursor—and erase text. But ncurses does little at present with those, since (aside from XTerm), most terminal emulators support those features poorly or not at all.

The ncurses library for the most part assumes that **bce** means the particular set of choices made for Linux console and xterm. Not all popular terminals match those choices. The standard is silent on "correct" behavior. There are two ways that ncurses may handle these differences:

1. provide a terminal description which omits the specific capability which differs from the assumption, or
2. in the library itself, be more pessimistic in optimizing the given capabilities.

The former is preferred, of course. The library should not be cluttered up with special cases.

My xterm program doesn't recognize color

Another specific problem lies with the terminfo description xterm. There are several xterm- and xterm-like terminfo entries in ncurses' terminfo database, corresponding to different terminal emulators. Only one is named "xterm", and that corresponds to the standard one. A fresh install of ncurses provides a choice between the current and previous standard ones:

xterm-new
: that is based on XFree86 xterm (plus updates since 2006, of course). See xterm's change log for details.

xterm-old
: This is the same as `xterm-r6`, e.g., the X11R6 xterm. That program (X11R6 xterm) does not support ANSI text color.

For either flavor, your packager may have customized the definitions for backspace and delete to match the conventions of your system.

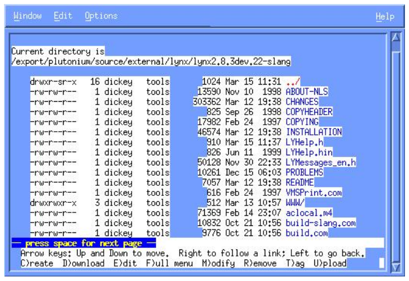
A fresh install of xterm on top of ncurses installs its terminfo entry as "xterm".

Why doesn't ncurses use $COLORTERM?

$COLORTERM is an environment variable used by some applications developers who are constructing programs that run on `rxvt`, a terminal emulator, using the `slang` library. It tells the `slang` library to ignore the terminal description, using a set of built-in capability strings which produce ISO 6429 (aka "ANSI") color on that emulator. The choice of capability strings may work for other emulators, but in general does not (e.g., terminals which lack the back color erase capability, such as Tera Term, CDE dtterm and nxterm).

Viewed as a fallback, $COLORTERM is perhaps acceptable (ncurses can be configured with built-in predefined terminal descriptions), but as a modifier to existing terminal descriptions it only leads to confusion, since most emulators that support color differ in minor details from the model which is supported by `slang`.

For example, dtterm nominally emulates a DEC vt220. Someone who knows this, but is also told that it supports color may (based on the usual misinformation available in newsgroups) try setting $TERM to "ansi", "linux" or "xterm-color". (Use `infocmp` to see why this is uniformly bad advice). This figure illustrates what goes wrong when following that advice:



```
Current directory is
/export/plutonium/source/external/lynx/lynx2.8.3dev.22-slang

   drwxr-sr-x   16 dickey   tools        1024 Mar 15 11:31 ../
   -rw-rw-r--    1 dickey   tools       13590 Nov 10  1998 ABOUT-NLS
   -rw-r--r--    1 dickey   tools      303362 Mar 12 19:38 CHANGES
   -rw-rw-r--    1 dickey   tools         825 Sep 26  1998 COPYHEADER
   -rw-rw-r--    1 dickey   tools       17982 Feb 24  1997 COPYING
   -rw-r--r--    1 dickey   tools       46574 Mar 12 19:38 INSTALLATION
   -rw-r--r--    1 dickey   tools         910 Mar 15 11:37 LYHelp.h
   -rw-rw-r--    1 dickey   tools         826 Jun 11  1999 LYHelp.hin
   -rw-rw-r--    1 dickey   tools       50128 Nov 30 22:33 LYMessages_en.h
   -rw-rw-r--    1 dickey   tools       10261 Dec 15 06:03 PROBLEMS
   -rw-r--r--    1 dickey   tools        7057 Mar 12 19:38 README
   -rw-rw-r--    1 dickey   tools         616 Feb 24  1997 VMSPrint.com
   drwxrwxr-x    3 dickey   tools         512 Mar 13 10:57 WWW/
   -rw-rw-r--    1 dickey   tools       71369 Feb 14 23:07 aclocal.m4
   -rw-rw-r--    1 dickey   tools       10832 Oct 21 10:56 build-slang.com
   -rw-rw-r--    1 dickey   tools        9776 Oct 21 10:56 build.com
-- press space for next page --
  Arrow keys: Up and Down to move.  Right to follow a link; Left to go back.
  C)reate  D)ownload  E)dit  F)ull menu  M)odify  R)emove  T)ag  U)pload
```

There is a `dtterm` terminfo entry which provides correct behavior.

Why not just use TERM set to "xterm-color"?

ncurses has a terminal description named `xterm-color`. Users assume that means it will work properly for "any" *xterm*. That is incorrect. It combines the simplest form of ANSI colors with the older X11R6 xterm. Originally, `xterm-color` corresponded to the *color_xterm* from the mid-1990s. That was superseded by XFree86 xterm in 1996. That is better than nothing, however using it in modern xterm (or anything accurately claiming to be compatible), these problems would occur:

- selections from colored backgrounds will add unnecessary spaces to the pasted text.
- the function keys do not necessarily match (F1-F4 for instance), and will be incomplete (for example keys modified with shift-, control-).
- scrolling will be noticeably slower.
- some visual features (flash, hidden cursor) are not available.
- it will not work properly in `luit`.

Some terminal emulators may set this value; however it is unlikely that any current emulators implement this particular set of limited features. It is more likely that a more capable description exists or (given suitable documentation) that one could be constructed.

For instance, it has been reported that Mac OS X's bundled terminal emulator uses this value. However, reliable reports of its actual capabilities say that there are differences, which are addressed in ncurses as the `nsterm` entry. In this case, infocmp shows

```
comparing xterm-color to nsterm.
    comparing booleans.
        hs: F:T.
        km: T:F.
        npc: F:T.
        xon: F:T.
    comparing numbers.
        colors: 8, 16.
        ncv: NULL, NULL.
        pairs: 64, 256.
        wsl: NULL, 50.
    comparing strings.
        acsc: '``aaffggiijjkkllmmnnooppqqrrssttuuvvwwxxyyzz{{||}}~~', '``aaffggjjkkllmmnnooppqqrrssttuuvvwwxxy
        blink: NULL, '\E[5m'.
        civis: NULL, '\E[?25l'.
        clear: '\E[H\E[2J', '\E[H\E[J'.
        cnorm: NULL, '\E[?25h'.
        dsl: NULL, '\E]2;\007'.
        el1: NULL, '\E[1K'.
        enacs: '\E)0', '\E(B\E)0'.
        flash: NULL, '\E[?5h$<200/>\E[?5l'.
        fsl: NULL, '^G'.
        hpa: NULL, '\E[%i%p1%dG'.
        ich: NULL, '\E[%p1%d@'.
        ich1: NULL, '\E[@'.
        invis: NULL, '\E[8m'.
        is2: '\E[m\E[?7h\E[4l\E>\E7\E[r\E[?1;3;4;6l\E8', NULL.
        ka1: NULL, '\EOq'.
        ka3: NULL, '\EOs'.
        kb2: NULL, '\EOr'.
        kbs: '^H', '\177'.
        kc1: NULL, '\EOp'.
        kc3: NULL, '\EOn'.
        kend: NULL, '\E[F'.
        kent: NULL, '\EOM'.
        kf1: '\E[11~', '\EOP'.
        kf18: '\E[32~', '\E[22~'.
        kf2: '\E[12~', '\EOQ'.
        kf3: '\E[13~', '\EOR'.
        kf4: '\E[14~', '\EOS'.
        kfnd: '\E[1~', NULL.
        khome: NULL, '\E[H'.
        kich1: '\E[2~', NULL.
        kmous: '\E[M', NULL.
        kslt: '\E[4~', NULL.
        meml: '\El', NULL.
        memu: '\Em', NULL.
        op: '\E[m', '\E[0m'.
        rmam: NULL, '\E[?7l'.
        rs2: '\E[m\E[?7h\E[4l\E>\E7\E[r\E[?1;3;4;6l\E8', '\E>\E[?3l\E[?4l\E[?5l\E[?7h\E[?8h'.
        setab: '\E[4%p1%dm', '\E[%?%p1%{8}%<%t%p1%'('%+%e%p1%{92}%+%;%dm'.
        setaf: '\E[3%p1%dm', '\E[%?%p1%{8}%<%t%p1%{30}%+%e%p1%'R'%+%;%dm'.
        setb: NULL, '%p1%{8}%/%{6}%*%{4}%+\E[%d%p1%{8}%m%Pa%?%ga%{1}%=%t4%e%ga%{3}%=%t6%e%ga%{4}%=%t1%e%ga%{6}
        setf: NULL, '%p1%{8}%/%{6}%*%{3}%+\E[%d%p1%{8}%m%Pa%?%ga%{1}%=%t4%e%ga%{3}%=%t6%e%ga%{4}%=%t1%e%ga%{6}
        sgr: NULL, '\E[0%?%p6%t;1%;%?%p2%t;4%;%?%p1%p3%|%t;7%;%?%p4%t;5%;%?%p7%t;8%;m%?%p9%t\016%e\017%;'.
        sgr0: '\E[m', '\E[m\017'.
        smam: NULL, '\E[?7h'.
        tsl: NULL, '\E]2;'.
        vpa: NULL, '\E[%i%p1%dd'.
```

Additionally, Mac OS X 10.7 is reported to use `xterm-256color` as a default `$TERM` value. This differs from `xterm-color` in several ways, in particular, the support for bce. It also differs from the recommended `nsterm-256color` (infocmp reports 111 differences).

Why not just use TERM set to "xterm"?

It is easy to find people recommending just setting `$TERM` to "xterm", noting that several developers have used that as a default value for their terminal emulators. However, this is usually (except for xterm itself) a bad idea. The reason that those terminal emulators use "xterm" is not because they actually match the behavior, but

- it seems too much work to install the correct terminal definition on each machine (this was the reason given for rxvt's having done so for several years), or
- it seems inconvenient to the user to put up with the absence of a given terminal description when making remote connections (which pass `$TERM` to the remote system).

However, "xterm" refers to a specific terminal emulator. Some of its details are widely emulated, while others (such as the function key definitions) are not.

Ncurses has provided accurate descriptions for the different terminal emulators for many years. While each terminal may implement escape sequences which are not in the others (none implement as much as half of xterm), what is important for curses applications is what can be represented in the terminal description. Using infocmp, here is the amount of difference seen by ncurses for commonly used terminal emulators as of January 2012:

Counting differences from xterm with infocmp

Description	Count
ansi (for reference)	137
konsole	58

linux	123
mlterm	59
nsterm (Apple Terminal)	117
putty	128
rxvt	124
screen	107
screen.xterm-new	11
vte (e.g., gnome-terminal)	55
xterm-color	112
xterm-new (current standard)	0
xterm-old (old standard)	115
vt100 (for reference)	154

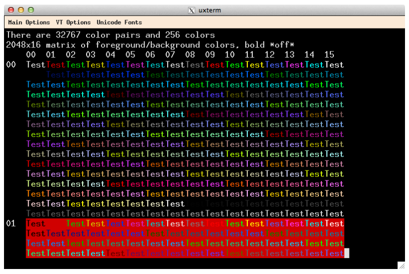
Noting that infocmp lists 182 capabilities for xterm-new, entries with more than a small number of differences demonstrate lack of compability of xterm "me-too's" versus xterm itself.

Why not make "xterm" equated to "xterm-256color"?

It breaks things, in more than one way. Seriously, no one should be asking this question, but see [Why not just use TERM set to "xterm"?](). Still (writing in mid-October 2012), I see that the answer is needed.

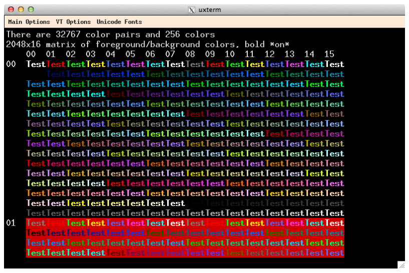
First, some background is needed. The xterm 256-color feature started with Todd Larason's patch, which I added in mid-1999 in [patch #111](). Though we (including Steve Wall's changes for 88-colors) continued to refine the feature for some time, it was not at first used much by applications. For instance, though I added the terminal description to ncurses, it was only partly supported by ncurses until I added the extended color features beginning in [January, 2005]().

Here are screenshots showing the ncurses test-program with extended colors, using menu entry "C", added in that time period. The first uses normal-weight text:



```
There are 32767 color pairs and 256 colors
2048x16 matrix of foreground/background colors, bold *off*
    00  01  02  03  04  05  06  07  08  09  10  11  12  13  14  15
00  TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
        TestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
01  Test    TestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
```

and the second uses bold font:



```
There are 32767 color pairs and 256 colors
2048x16 matrix of foreground/background colors, bold *on*
    00  01  02  03  04  05  06  07  08  09  10  11  12  13  14  15
00  TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
        TestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
01  Test    TestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
    TestTestTestTestTestTestTestTestTestTestTestTestTestTestTestTest
```

Other terminal emulators adapted the feature before any noticeable use by applications. After noticing these and verifying their behavior, I added corresponding entries to ncurses' terminfo.src file. Here are relevant changes:

88/256-color Terminal Descriptions

Date	Change
1999/11/27	add xterm-88color, xterm-256color
2002-06-22	add rxvt-16color, ibm+16color
2005-01-29	update pairs for xterm-88color and xterm-256color to reflect the ncurses extended-color support
2005-02-26	add aixterm-16color to demonstrate 16-color capability
2006-02-18	add nsterm-16color entry
2006-04-22	add xterm+256color building block, and add gnome-256color, putty-256color, rxvt-256color
2007-07-14	add konsole-256color entry

2008-08-23	add Eterm-256color, Eterm-88color, and rxvt-88color
2009-10-03	add linux-16color; add ccc and initc capabilities to xterm-16color
2010-02-06	add mrxvt-256color
2010-06-12	add mlterm+256color entry
2012-03-31	correct order of use-clauses in st-256color
2012-08-11	add nsterm-256color, make this the default nsterm

Things started to change around 2005-2006 as you can see by looking at the dates in the table. Red Hat #175684 gives some clue, stating that the "next version of Emacs has support for 256 color terminals". Emacs on the console by the way uses termcap, not terminfo and certainly not (n)curses.

I chose to add new entries rather than change xterm (and others) because they are incompatible. There is more than one aspect to consider:

- some applications on the same machine will blindly attempt to use all of the colors without being ready to handle more than 8/16 colors. There was for example more than one bug-report with Fedora of this type. The symptom was that text would be invisible (black on black). The bug reports were eventually closed without correcting the problem.
- connections to/from remote machines with a different notion of *xterm* will behave inconsistently.
- termcap-style (or the rare terminfo-style) applications on a given machine are the only ones that can use the -256color or -88color flavors. But aliasing those to *xterm* will break ncurses applications.
- there is already a widely-distributed terminal entry with the desired features. Since it has proven to be inconvenient to use the *xterm-256color* name everywhere, renaming it to *xterm* will simply shift the problem without providing a solution.
- making incompatible changes to a widely-used terminal description produces bug-reports. I have limited changes of that type in xterm to describing new features, not changing existing features. Even with that, there have been complaints for each change. Consider

  SGR 22
  : rant by an end-user, twelve years after the change in question (viewer discretion advisory). David Dawes was the first to point out that this change introduced a problem with remote systems. After some discussion we decided to keep it.

  Function-key modifiers
  : In xterm's patch #94 I added function-key modifiers. In patch #167 I modified the protocol to address a problem with Emacs (though making it configurable). In the meantime, KDE and GNOME had copied the earlier feature into their terminal emulators. Ten years later, neither has updated to the 2002 version of xterm's function-keys. There were several reports, the first that I noticed appeared in Red Hat's system. For example see #127773. Ubuntu #96676 is another example (still unresolved in 2012).

  Meta-mode
  : Debian #444250. Essentially the problem here was that adding a flag to the description telling how to enable/disable xterm's meta mode exposed a differing interpretation of meta mode in bash which dates back to around 1990.

Although ncurses (starting with release 5.5 in 2005) can be configured to support 256 colors, it seems that no widespread distribution is using the extended colors feature (certainly not Red Hat or Debian — perhaps OpenSuSE does). That (and the extended mouse feature) are the main features for the "ABI 6", which could easily coexist with the usual "ABI 5" ncurses.

Without widespread use of ABI 6, there is no reason to change the primary flavor of any of ncurses' terminfo entries to the -256color or -88color flavor.

In the Debian distribution there was an attempt in 2008 on the part of the ncurses package maintainer to provide a migration path to ABI 5. That effort died, and does not at this time have a clear successor.

Just because it is not viable does not prevent others from assuming it is. For example:

- Debian #230990 asks for the extended mouse support. It was opened in February 2004.
- Debian #405602 took 18 months to move the xterm-256color entry from the ncurses-term to the ncurses-base package in mid-2008.
- Debian #532537 asks for a change of *xterm* to *xterm-256color*. It was opened in June 2009. As justification, it stated that "all Debian systems have been supporting the 256-colour interface for as long as anyone can remember". The package changelog for xterm states that the feature was added in January 2006, which when compared with the Debian release cycle makes it very far from "all Debian systems".
- In mid-2012, I noticed that Fedora 18 would have *xterm* aliased to *xterm-256color*. Some notes are here. However, the proponents of the change place too much emphasis on just changing it and pay less attention to the ensuing problems.

Why only 16 (or 256) colors?

Before getting to the point, some background is needed.

Curses applications run in terminals (or terminal emulators). They set colors for the screen and parts of it by sending additional characters (escape sequences) to the screen.

The ANSI standard described 8 colors, did not in any sense allude to more colors. This is a palette-based set of colors; there is no standard for how those colors are composed.

A few terminals recognize the **aixterm** 16-color extension (not part of any standard), e.g., xterm patch #39. Again, there is no agreement between different terminals on the actual colors used. (While xterm is configurable, some of the others are not).

Starting with xterm patch #111, there is the "xterm-256color" terminal type. Using SGR 38 and 48 in an arguably nonstandard manner (at the time there was no free access to ISO 8613-6), it provides a predefined palette of 256 colors which can be modified through escape sequences. With the terminals which imitate this (none matches xterm's behavior exactly), there are two problems:

- the colors differ
- the interpretation of the escape sequences is generally incomplete, e.g., not recognizing the sequences for modifying the palette or not accepting only one color per escape sequence.

These deficiencies are well known, easily demonstrated.

Sometime after xterm patch #111, a freely accessible version of ISO 8613-6 (ITU T.416) was made available. Some (mostly secondhand) discussion of the ramifications of this is in KDE #107487.

A small minority of terminals support an additional interpretation of SGR 38 and 48 alluded to in ISO 8613-6 as *direct colour in RGB*. The term "direct colour" has been interpreted as "24-bit mapping of RGB" (it is an interpretation, since the ISO standard referenced predated any contemplated implementation which matched this feature).

As is often the case with the ISO terminal standards, there is little or no prior art to use as a basis for standardization of these features. Accordingly, the existing implementations differ:

- ISO 8613-6 prescribes a different syntax than that used for xterm patch #111 as the *indexed colour* feature.
- Konsole used the xterm syntax for its *256-color* and *direct color* features, as noted in a README file.
  The example script, by the way, demonstrates the former (*256-color* feature) and was copied/renamed from xterm's sources, as shown here. There is no corresponding script for *direct colour* in Konsole's source repository.
  The *direct color* feature was largely ignored for a few years. There were occasional comments, such as this on the vim-users discussion group.
- Some attempts were made by GNOME developers to apply similar changes to other terminals; no coherent description has been found summarizing the success of that endeavor.
- After some reflection, I provided in xterm (with patch #282) support for the direct color feature. Both that and the earlier 256-color feature are usable with either the ISO 8613-6 syntax, or the xterm patch #111 syntax.
- At this writing (January 2014), none of the other terminals which I have tested accepts the ISO 8613-6 syntax (there are some unverified reports).

Beyond the issue of escape sequences, konsole and xterm use different library calls for implementing *direct color*. This makes no difference to an application which might use those escape sequences, but bears some comment:

- konsole uses **Qt**. In its source code (see xxx), it stores the color information in a QColor class. The documentation mentions

  ```
  QColor is platform and device independent. The QColormap class maps the color to
  the hardware.
  ```

  Because konsole's source code does not refer explicitly to QColormap, it relies on default behavior of the class.
  The documentation for QColormap gives half a clue, in its comment about modes. Qt's documentation is vague in many respects (in sharp contrast to MSDN), leaving developers no recourse other than reading its source code.
  Here is a starting point: Qt 4.8 src/gui/painting.

- In the 4.8 branch, we find that the QColormap class for X11 was doing essentially what xterm does, i.e., color maps.
- In the *5.0* source (still in development at the time the *vim_use* discussion took place—released 6 months later), there are some changes for QColormap. The class initialization distinguishes between 24-bit and 8-bit color. The X11-relevant details have been moved away from this class.
- Distinguishing between 8-bit and 24-bit color is a useful simplificaton for X11, since it trades off between storing an 8-bit value in one of the columns for the R/G/B structure versus storing three 8-bit values.
  XTerm checks for this special case to allow exact color matches. Otherwise, it uses the closest color match.
- Because xterm allows the application to set the palette which is used for either style of color (256-color or direct), providing "lot of colors" would have been pointless.
- Konsole has no provision for modifying the palette, in either case.

The implication of the *direct colour* feature is "lots of colors". On the surface, that may seem attractive. However, there are several considerations:

- who will use it?
- how will they use it?
- when will they use it?
- where will they use it?
- how many colors do they need?
- what are the performance tradeoffs?

To answer the question, bear in mind that this is an area of special interest to only a handful of developers. End-users read “16 million colors” and uncritically accept arguments that they can effectively use this. However, there are only (at most) a few thousand characters on a terminal's screen at a time. That number of colors exceeds by a couple of orders of magnitude the ability of anyone to discern the differences and rely on those distinctions to aid them in viewing text.

Moving past the first couple of points, the availability of *direct color* depends on how the application uses the information. As a **terminfo** description, that works for applications which use the terminal database directly. The principal ones which come to mind are text editors (emacs and vi-clones). Generally their developers have chosen to make their task more difficult by remaining with **termcap** (which cannot express the relevant escape sequences). So they incorporate some hard-coded behavior. Working through these obstacles, it is possible to “use” termcap to provide “lots of colors” without affecting the ncurses library. Termcap applications do not use much of ncurses; nor do their developers contribute significantly to ncurses.

Some mailing-list comments indicate that the existing capabilities are not well-understood by developers, e.g., this discussion in October 2013. Taking the discussion into account, consider the linux-c terminal description dating from 1996. That sets up a palette using a color index value mapped to R/G/B components. Other longstanding uses in ncurses include these:

linux-c-nc, putty, xterm-16color, and xterm+256color

Some additional comments on mapping colors using `initc` and `initp` are found in this discussion.

Applications which *use* ncurses are a different matter. Unlike termcap applications, curses libraries (and analogous ones such as s-lang) use combinations of the terminal capabilities to allow their applications to work on a wide variety of terminals. There are a few limitations, due to the early implementation of terminfo:

- 4096 size-limit on compiled entries
- 16-bit limit for numbers (to keep them small), which are all *signed* values
- Because some terminal types did not provide a way to set foreground and background colors independently, the concept of *color pairs* was introduced.

Using the ncurses 5 ABI, you have available 16 colors, or 256 pairs of colors. Using the ncurses 6 ABI, you would have 256 colors, or 32767 pairs (the limit for a signed 16-bit number). That limit is good enough for realistic applications, which could not have that many character cells on a screen simultaneously (unless of course, using 1-pixel fonts to pretend to draw *graphics*, e.g., AA-lib). Since ncurses is used to draw *text*, that is not a valid issue.

S-lang does not implement color-pairs (or padding, etc.), However (reading the source code), it also is limited by the terminal descriptions, using a maximum color index value 32767 (*15*-bits—signed short). No method for changing 15 into 24 was noted in this discussion. However, that leads to a bug report with a patch which isn't fully "ABI compatible".

No matter how one uses the limits based on the terminal database, scaling is needed to map onto the actual device.

Bypassing the limits which can be derived from the terminal database (and veering off into the swamp of hard-coded behavior), the idle spectator might ask “why not add a special ad hoc interface to (somehow) just do it?” The problem is that you cannot “just do it”—both ncurses and s-lang do optimization to address their legitimate users (those who are using the superior performance due to working with characters, especially via remote connections). They have longstanding APIs which expose some of the related information. Adding an ad hoc interface will either break compatibility, or require duplicating information. Either way, most of the existing users would be adversely affected.

Some hint of this is given in GNOME #704449. Given the source of the comment, the effect is understandably understated. The usual reason given for wanting this feature has been for making pretty color schemes for text editors (e.g., this and this).

This item was prompted by a recent advertisement posted to the ncurses mailing list. Beginning with a fallacious premise (using a source known to be erroneous), its author *selects* specific instances in an attempt to persuade the reader that a majority of terminals implement TrueColor (“16 million colors”). Without significant revision, there is nothing more to discuss.

Ncurses resets my colors to white/black

This is the way SVr4 curses works. From the XSI Curses specification

```
The start_color() function also restores the colors on the terminal to terminal-
specific initial values. The initial background color is assumed to be black for
all terminals.
```

If your terminal description does not contain the `orig_pair` or `orig_colors` capability, you will be unable to reset colors to the pre-curses state. This happens, for example, with `aixterm`.

However, if your terminal does support resetting colors to a default state, ncurses will use this when exiting Curses mode. Terminal types that do this include the Linux console, `rxvt` and the XFree86 `xterm`.

Ncurses 4.1 provides an extension `use_default_colors()` which allows an application running on a terminal which supports resetting colors to mix the default foreground and background colors with the 8 defined curses colors.

Why are red/blue interchanged?

You may notice if you are porting an application from SVr4 curses to ncurses (or the other way), that some versions of ncurses have some pairs of colors interchanged with respect to SVr4 curses. This is a bug in ncurses (sorry). Someone made an error translating terminal descriptions, and confused the setaf/setab terminal capabilities with the setf/setb capabilities.

The 8 colors black, red, green, yellow, blue, magenta, cyan, white are coded in the ANSI (setaf/setab) convention with red=1, green=2 and blue=4, while the older (setf/setb) uses red=4, green=2 and blue=1. SVr4 curses accommodates either, interchanging colors in the setf/setb to match the setaf/setab style. Ncurses' terminfo database incorrectly renamed the setaf/setab capabilities to setf/setb, making it incompatible with the SVr4 curses library.

This was corrected in ncurses 4.1, but incorrect in all preceding versions.

Other Display Problems

Line-drawing characters come out as x's and q's

The x's and q's correspond to a table (from terminfo/termcap) which tells ncurses how to map the "alternate" character set to the terminal's set of graphic characters. The reference for this table comes from the vt100. If the unmapped characters appear, then the terminal emulator does not recognize the escape sequence for switching between normal and alternate fonts that is given in the terminfo description.

There are several cases of note:

- Terminal emulators which use a different escape sequence or different range for mapping the resulting characters. For instance the so-called vt100-compatibles such as Linux console and Tera Term.
- Terminal emulators which are locale-sensitive. Again, Linux console is a problem area when running in UTF-8 mode, since its nominal vt100-compatibility is further lessened by ignoring the escape sequences dealing with fonts. The `screen` utility also has the same problem; whether to make the implementation simple or to copy the Linux console. It ignores vt100-style font switching when the locale is a UTF-8 flavor.
- If you happen to be using Solaris, it is often configured to prefer its terminal database to ncurses, even when ncurses is installed. However, its terminal description for xterm omits the `enacs` which is used to enable line-drawing. This does not work well with applications such as `screen` and `luit`.

For the first case, you simply have to find the correct terminfo description. Fixing the latter is harder, since the damage is done outside ncurses. (Though one can easily make things compatible enough that this particular issue would never appear, that style of solution is not deemed proper by some coders).

The normal ncurses libraries support 8-bit characters. The ncurses library can also be configured (`--enable-widec`) to support wide-characters (for instance Unicode and the UTF-8 encoding). The corresponding wide-character `ncursesw` libraries are source-compatible with the normal applications. That is, applications must be compiled and linked against the `ncursesw` library.

The ncurses 5.3 release provides UTF-8 support. The special cases of Linux console and `screen` were addressed in development patches completed at the end of 2002.

Line-drawing characters come out as Latin-1 characters

Those capital A's with dots on top. Yes, that is almost always a mismatch with the terminfo description.

There are some special cases such as Tera Term which are related to the font.

Line-drawing characters do not appear

There is no character at all where the line-drawing should appear, and other characters may be shifted around the screen. This may happen on the Linux console, but also on some other terminal emulators. The line-drawing characters do not appear because the terminal emulator is treating them as illegal characters.

ncurses starts by assuming that the terminfo is correct, and overrides it for some special cases which are known to misbehave. See the discussion of `NCURSES_NO_UTF8_ACS` in the ncurses manpage for details.

Why does my cursor blink in Emacs?

Depending on Emacs, or other programs, the answer differs. Most often this question is raised by people using the Linux console.

There are three capabilities defined in terminfo which affect the cursor:

Terminfo Cursor Capabilities

Full name	Terminfo	Termcap	Description
cursor_invisible	civis	vi	make cursor invisible
cursor_normal	cnorm	ve	make cursor appear normal (undo civis/cvvis)
cursor_visible	cvvis	vs	make cursor very visible

The three capabilities appear in the terminfo database 128, 178 and 103 times respectively. Some of those terminfo entries are building blocks. For the entries reported by toe, there are 670 which modify the cursor, out of 1629 entries. Some of these allow only between invisible/normal or normal/visible. There are 159 which allow all three possibilities, as well as 27 which allow only one choice.

The cursor blinks because the application (Emacs for instance) uses the `cvvis` capability. "Very visible" for a terminal cursor generally means that it blinks. Various tradeoffs are possible, depending on the terminal between underline, block, blinking.

Applications modify the cursor either via curs_set (in ncurses) or by sending the appropriate escape sequence to the terminal using termcap or terminfo calls. While `curs_set` refrains from substituting one escape sequence for another missing one, this is not true of termcap/terminfo applications. Frequently they replace a missing `cvvis` with an available `cnorm`. After a while, the programmers forget the difference, and surprise their users when someone upgrades a terminal description.

The Linux console tie-in happened with ncurses in 1999, by adding `cvvis` to the "linux" terminal description.

Outside of Emacs, the replacement of a missing `cvvis` with `cnorm` is the most common cause of surprise.

But Emacs is different. It intentionally uses the *"very visible" blinking cursor* by default. There is documentation here. The earliest available version of the code from 1991 uses `TS_visual_mode` in this way.

Keyboard Problems

My cursor keys do not work

For instance, you may press the up-arrow and see "OA" or "[A", rather than having your program recognize the up-arrow.

The terminal description may be correct, or the program may not be making proper use of it. ncurses (and other programs which use terminal descriptions) simply match the characters sent by the terminal against a list of strings defined in the terminal description.

vt100's can send different strings for cursor-keys (and keypad-keys) depending on how they are initialized. Terminal emulations which provide vt100 functionality behave the same way. There are two modes:

normal mode
: This corresponds to the power-up state of the terminal. Cursor keys send escape sequences (strings beginning with the ASCII escape character) that mimic cursor-movement control sequences. Those begin with `escape [`. An up-arrow is `escape [ A` (also expressed as `CSI A`.

application mode
: This is the programmed mode of the terminal. Cursor keys send escape sequences that can be easily distinguished from cursor-movement control sequences. Those begin with `escape O`. An up-arrow is `escape O A` (also expressed as `SS3 A`.

The terminal description describes only one mode. Usually that is the application mode.

The terminal description tells how to initialize the terminal, e.g., to make the cursor-keys send application mode escape sequences. Long ago, when there were many types of terminals in use, some would send nothing for special keys unless the terminal was initialized in this way. To accommodate all types of terminals, the convention was that a program would initialize the cursor keys when it needed them. That works well for full-screen programs. Terminal descriptions (usually) describe the application mode for this reason.

It does not work well for command-line programs, which do not initialize the terminal. Keys which differ between normal and application modes are not available for programs which do not initialize the terminal.

For example, the `readline` library (which was originally hardcoded, and did not use termcap or terminfo), exploited the feature of vt100's that their cursor-keys transmit something in normal mode.

The `readline` library (which is used by `bash`) did this for several years before someone connected it to a termcap library. Around that point, someone made a terminal description for the Linux console which "solved" the problem of initializing the terminal for `bash` by not initializing it at all. Cursor keys always send normal-mode strings with this arrangement.

Granted, that works—provided that the terminal is initialized consistently. If your terminal happens to "solve" the problem in a different way, e.g., by starting in application mode, and if your environment sets `TERM` to "linux", you may have problems with your cursor-keys.

Alt-keys do not work in bash

Depending on the application, users expect an "alt" key to do one of two things:

- send an escape character before special keys such as cursor-keys, or
- act as an extended shift, allowing you to enter codes for Latin-1 values from 160 to 255.

xterm supports both features, e.g., with the `eightBitInput` and related resources.

Bash users have an extra problem in xterm when using a recent terminal description (since xterm patch #216). For completeness, I noted in the terminal description how an application can set/reset "meta" mode. It turns out that bash turns on meta mode if it is available. It is not an optional feature in bash; bash's developers assumed that you would use it. That is an odd assumption, since the feature is rarely implemented. There are (counting xterm) only three terminal descriptions in ncurses' terminal database providing this (the low-level blocks for Ann Arbor, SCO, and XTerm).

Making it configurable was proposed in this bug report. Earlier, bash's maintainer had expressed reluctance to address the issue. However, bash 4.1 added an "enable-meta-key" feature, dated 9 October 2009. That makes it possible for users to disable it.

My home/end keys do not work

This is usually due to an incorrect terminal description. When it is not, it is due to the terminal emulator itself.

Application developers tend to hardcode references in their programs to the "home" and "end" key, forgetting that the presence of any particular function key depends on the terminal.

Part of the problem with terminal descriptions is due to the way xterm's terminal descriptions are used in ncurses. Because xterm emulates a VT220, the terminal descriptions provided with it follows the DEC keyboard convention, which does not provide home/end keys. Rather, it provides find/select keys. See for example the discussion of function keys and keypad.

On the other hand, the descriptions in ncurses reflect the Sun and PC keyboards which many people use. They define home/end keys for xterm.

However, some packagers for ncurses have pasted xterm's terminfo descriptions into the ncurses distribution. The effect is to remove the home/end key definitions from ncurses.

The problem is easily fixed by ignoring the packager's improvements and reinstalling the ncurses terminfo database. This section tells where to get it.

How can I use shift- or control-modifiers?

The standard response is "curses doesn't do that".

There are workarounds. Some explanation is needed first.

Most implementations of curses work with terminals that use serial communication. Generally those were inexpensive. Adding keys to the keyboard was probably less expensive than adding logic to handle different types of modifiers (such as shift). Terminals that could send different types of modifiers used to be rare.

However, the IBM PC provided a widely available platform with a keyboard that could provide modifier information. On the other hand, there was no standard protocol for sending the information on a serial line.

Most of the activity in exploiting this platform consisted of ad hoc implementations for computer consoles, such as for SCO, OS/2, etc., which associated the control-, shift- and alt-key modifiers with function-key numbering. That is, rather than attempting to extract information on the modifiers from function key expressed as a character string, the various combinations are numbered (using function-key number, multiplied by a code corresponding to the modifiers) and associated with the numbered function keys in the termcap/terminfo description.

For example, the `scoansi` description encodes the modifiers by an arbitrary sequence from the possible final characters for an ANSI control, e.g.,

```
normal
     kf1=\E[M, kf10=\E[V, kf11=\E[W, kf12=\E[X, kf2=\E[N,
     kf3=\E[O, kf4=\E[P, kf5=\E[Q, kf6=\E[R, kf7=\E[S, kf8=\E[T,
     kf9=\E[U,
F13-F24 are shifted F1-F12
     kf13=\E[Y, kf15=\E[a, kf16=\E[b, kf17=\E[c, kf18=\E[d,
     kf19=\E[e, kf20=\E[f, kf21=\E[g, kf22=\E[h, kf23=\E[i,
     kf24=\E[j,
F25-F36 are control F1-F12
     kf25=\E[k, kf26=\E[l, kf27=\E[m, kf28=\E[n, kf29=\E[o,
     kf30=\E[p, kf31=\E[q, kf32=\E[r, kf33=\E[s, kf34=\E[t,
     kf35=\E[u, kf36=\E[v,
F37-F48 are shift+control F1-F12
     kf37=\E[w, kf38=\E[x, kf39=\E[y, kf40=\E[z, kf41=\E[@,
     kf42=\E[[, kf43=\E[\\, kf44=\E[], kf45=\E[\^, kf46=\E[_,
     kf47=\E[`, kf48=\E[{,
```

Some combinations are missing (kf14 for instance corresponds to a back-tab).

A different scheme is used by `rxvt` (which was influenced by xterm and different hardware portability tradeoffs):

```
kf1=\E[11~, kf10=\E[21~, kf11=\E[23~, kf12=\E[24~,
kf13=\E[25~, kf14=\E[26~, kf15=\E[28~, kf16=\E[29~,
kf17=\E[31~, kf18=\E[32~, kf19=\E[33~, kf2=\E[12~,
kf20=\E[34~, kf3=\E[13~, kf4=\E[14~, kf5=\E[15~,
kf6=\E[17~, kf7=\E[18~, kf8=\E[19~, kf9=\E[20~,
kf21=\E[23$, kf22=\E[24$,
kf23=\E[11\^, kf24=\E[12\^, kf25=\E[13\^, kf26=\E[14\^,
kf27=\E[15\^, kf28=\E[17\^, kf29=\E[18\^, kf30=\E[19\^,
kf31=\E[20\^, kf32=\E[21\^, kf33=\E[23\^, kf34=\E[24\^,
kf35=\E[25\^, kf36=\E[26\^, kf37=\E[28\^, kf38=\E[29\^,
kf39=\E[31\^, kf40=\E[32\^, kf41=\E[33\^, kf42=\E[34\^,
kf43=\E[23@, kf44=\E[24@,
```

The pattern is not so obvious here. The developer who assigned the numbering chose certain combinations from a table which was too large to map into the available 60 numbered keys. Here is the complete table, noting that `F1` is the X11 code which is not necessarily synonymous with `kf1`:

Rxvt Function-Key Modifiers

X11 Key	Normal	Shift	Control	Shift+Control
F1	ESC [11 ~	ESC [23 ~	ESC [11 ^	ESC [23 ^
F2	ESC [12 ~	ESC [24 ~	ESC [12 ^	ESC [24 ^
F3	ESC [13 ~	ESC [25 ~	ESC [13 ^	ESC [25 ^
F4	ESC [14 ~	ESC [26 ~	ESC [14 ^	ESC [26 ^
F5	ESC [15 ~	ESC [28 ~	ESC [15 ^	ESC [28 ^
F6	ESC [17 ~	ESC [29 ~	ESC [17 ^	ESC [29 ^
F7	ESC [18 ~	ESC [31 ~	ESC [18 ^	ESC [31 ^
F8	ESC [19 ~	ESC [32 ~	ESC [19 ^	ESC [32 ^
F9	ESC [20 ~	ESC [33 ~	ESC [20 ^	ESC [33 ^
F10	ESC [21 ~	ESC [34 ~	ESC [21 ^	ESC [34 ^
F11	ESC [23 ~	ESC [23 $	ESC [23 ^	ESC [23 @
F12	ESC [24 ~	ESC [24 $	ESC [24 ^	ESC [24 @
F13	ESC [25 ~	ESC [25 $	ESC [25 ^	ESC [25 @
F14	ESC [26 ~	ESC [26 $	ESC [26 ^	ESC [26 @
F15 (Help)	ESC [28 ~	ESC [28 $	ESC [28 ^	ESC [28 @
F16 (Menu)	ESC [29 ~	ESC [29 $	ESC [29 ^	ESC [29 @
F17	ESC [31 ~	ESC [31 $	ESC [31 ^	ESC [31 @
F18	ESC [32 ~	ESC [32 $	ESC [32 ^	ESC [32 @
F19	ESC [33 ~	ESC [33 $	ESC [33 ^	ESC [33 @
F20	ESC [34 ~	ESC [34 $	ESC [34 ^	ESC [34 @

That is, the final character is again used to encode the modifiers. There are several terminals which use different final-character encoding schemes, e.g., dg (Data General), interix, and those based on `ansi.sys` or `cons25`.

xterm uses a different scheme, encoding the modifiers as a parameter. There are many combinations available. Here is a short example:

```
kf1=\EOP, kf13=\E[1;2P, kf25=\E[1;5P, kf37=\E[1;6P,
kf49=\E[1;3P, kf61=\E[1;4P,
```

All of that focuses on *function keys*. There are other special keys on the IBM PC keyboard, e.g., the editing keypad and cursor keys.

Here, conventional terminfo/termcap provides just a little help, but providing names for some of the shifted keys. xterm defines some of these:

```
kDC=\E[3;2~, kEND=\E[1;2F, kHOM=\E[1;2H, kIC=\E[2;2~,
kLFT=\E[1;2D, kNXT=\E[6;2~, kPRV=\E[5;2~, kRIT=\E[1;2C,
```

But xterm can also work for control-, alt-, and meta-modifiers. None of those are defined in conventional terminfo/termcap.

ncurses allows you to define extended descriptions, i.e., to make up your own names. The xterm terminfo descriptions do just that: they define names for the modified cursor- and editing-keypad keys which are just the xterm modifier code appended to the name for the shifted key. For example, the delete-key (kdc) can be represented as shown below:

Example of Xterm Extended Key (DC)

Xterm Code	Modifier	Extended Name
1 (or missing)	Normal	kdc
2	Shift	kDC
3	Alt	kDC3
4	Shift + Alt	kDC4
5	Control	kDC5
6	Shift + Control	kDC6

7	Alt + Control	kDC7
8	Shift + Alt + Control	kDC8

Other terminal descriptions can be (and in ncurses, have been) modified to use this naming scheme for extended keys:

- gnome-terminal and konsole use an older version of xterm's encoding (deprecated in 2002 because the parameter can be mistaken for a repeat count). Although the encoding is different (and these terminals omit some combinations), the functionality is similar.
- rxvt also provides distinct escape sequences for modified cursor- and editing-keys.

By adding these to the terminal description, curses applications will see the keycode rather than individual characters when processing keypad mode.

There are a few limitations though:

- the keycodes for the extended keys are generated at runtime, so the application must ask what their values are using key_defined.
- compiled terminfo descriptions, though larger than termcap's 1023 bytes, are still limited to 4096 bytes. Many applications (such as those built using `slang`) rely on this limit. Applications can work around this by defining keys at runtime, using define_key.

Using Hardware (Real) Terminals

Why does reset log me out?

This is a limitation of real hardware terminals: resetting them will break the communications to the host temporarily. Some terminal interfaces will tolerate this. Others (most) will interpret this as hanging up, and log you out.

The reset is specified in the terminal description, e.g.,

```
rs1=\Ec,
```

That is the hardware reset escape sequence for vt100. Some terminals provide enough control over their features that a very complicated substitute could be concocted for the normal reset which does not perform a hardware reset. In practice, this is not easily done.

Why do I get trash on the screen?

This is a problem of real hardware terminals. Cheap terminals and cheap interfaces do not do sophisticated flow control (e.g., XON/XOFF). Instead, they rely on a host which does not send them characters too rapidly. Remote terminal servers may provide flow control; direct console or serial port connections often do not. (If you are asking this question, you probably have inexpensive hardware).

Terminfo descriptions designed for these inexpensive terminals have delay times specified in the control sequences which take extra time, such as clearing the screen. For example, the vt100 description tells the application to wait 50msec after clearing the screen:

```
clear=\E[H\E[J$&lt;50&gt;,
```

Note: the `slang` library does not implement delay times, and is not suitable for applications which require direct connection to a hardware terminal. The author of that library states that no one uses hardware terminals any more, suggesting that I add this information to the FAQ.

Interaction with Other Programs

Redirecting I/O to/from a Curses application

In principle, you should be able to pipe to/from a curses application. However, there are caveats:

- Some (very old) curses implementations did not allow redirection of the screen. Ncurses, like Solaris curses, consistently writes all output to the standard output. You can pipe the output to a file, or use `tee` to show the output while redirecting.
- Ncurses obtains the screen size by first using the environment variables LINES and COLS (unless you have disabled it with the `use_env` call), then trying to query the output file pointer, and then (finally) the terminal description. If you are redirecting output, then a query based on the file pointer will always fail, resulting in the terminal description's size.
- Similarly, you can redirect input to an ncurses application. However, I have observed that the use of `setvbuf` (for better output buffering) interferes with the use of stream I/O on Linux-based systems (and possibly other platforms). Invoking `setvbuf` may (depending on the implementation) cause buffered stream input to be discarded. Ncurses does not use buffered input, however you may have an application that mixes buffered input with a curses session.

What about readline?

The `readline` library appeals to a number of people, who would like to use it within an ncurses application.

At first glance—but only for that instant—it appears that the library is flexible enough to substitute a different display driver, e.g., to output via something other than `tput()` and `putc()`. Close reading of its `display.c` file shows this is ultimately futile. Quoting (Brian Fox's comment from the early 1990s, still present in readline 6.1 in 2010) from that file gives some insight:

```
/* This is the stuff that is hard for me.  I never seem to write good
   display routines in C.  Let's see how I do this time. */
```

John Greco suggests a different approach, which discards ncurses' input functions (such as `wgetch`), using `rl_callback_read_char` to fill the `rl_line_buffer` array. That can be printed using ncurses.

Problems with Output Buffering

Ncurses shares with SVr4 curses a limitation which is documented in the C standard. To attain good performance, they buffer output data, e.g., with the `setvbuf` function (or equivalent, depending on the platform). The performance gain is noticable.

However, if your application spawns a subprocess, it will inherit the output stream from ncurses—still buffered. On several platforms this results in odd behavior, since normally the standard output is line buffered, making the output flushed at the end of each line. To solve this problem, your application should disable `setvbuf` before invoking the subprocess and restore it when resuming. That is, it should, but often cannot—that is the problem. The standard says that `setvbuf` must be called only after opening a stream and before performing any reads or writes with that stream.

If your application calls `initscr`, it uses the standard output, which ncurses assumes has not been written to, to which ncurses then applies buffering. (*Caveat*: The standard writers neglected to provide a mechanism for determining if the stream is indeed buffered). Adding a call to `setvbuf` to disable buffering may work or not. In at least one implementation, the C library continues using the buffer after the buffer is disabled, even if an `fflush` is first given. That is, it will produce a core dump.

The fix? Use `newterm` to initialize ncurses and manage the input and output streams yourself. For instance, you may simply open `/dev/tty` for input and output, and leave the standard input and output alone.

Mice and Windows

I can't cut/paste in xterm

This is a general FAQ relating to xterm. When an application sets xterm to any of its mouse tracking modes, it reserves the unshifted mouse button clicks for the application's use. Unless you have modified the treatment of the shifted mouse button events (e.g., with your window manager), you can always do cut/paste by pressing the shift key while clicking with the mouse.

Ncurses sets the mouse tracking mode as a result of your application's calls to `mousemask`, which is an extension.

Handling SIGWINCH (resize events)

It is possible to make an application *resize* when running in a windowing environment (e.g., in an `xterm`). This is not a feature of standard SVr4 curses, though some curses implementations (e.g., HP-UX) support this.

Within ncurses, there are two ways to accomplish this. One relies on side-effects of the library functions, and is moderately portable. The other is an extension to SVr4 curses.

- `endwin`/`refresh` when invoked will briefly exit curses and reinitialize the display, picking up the new screen size. Ncurses will reallocate the WINDOW data (e.g., curscr, stdscr) to reflect the new limits.
- `resizeterm` can be invoked directly to make ncurses resize its WINDOW data. I use it in my directory editor ded to achieve flicker-free resizing via a signal handler for SIGWINCH. (The documentation for HP-UX curses implies that they use a similar approach; I have been unable to make it work.)

Ncurses 5.0 can be configured to establish its own SIGWINCH hander. In this configuration, the `wgetch` function will return a special keycode `KEY_RESIZE` when a resizing event is detected. The signal handler also calls `resizeterm` (**Caveat**: `malloc` and `free` are not guaranteed to be safe for use in a signal handler).

There is a known problem using bash's *checkwinsize* misfeature. See Novell #828877, as well as the thread starting here on bug-bash, and of course the manpage for use_env. Shells which set `LINES` and `COLUMNS` might have been a good idea in 1995, but not as a rule more recently than that.

Linking with GPM (Linux console mouse library)

Ncurses works correctly with the Linux GPM (general purpose mouse) library. However, early on some distributors tampered with the library, making it not generally useful, by linking in a portion of the BSD curses library to satisfy references for `Gpm_Wgetch`. That prevented one from using ncurses' GPM support.

GPM provides limited support for xterm mouse control sequences. This is implemented in `GPM_Wgetch`, which makes some unreasonable assumptions about the curses library's internal behavior of `wgetch`. In particular, GPM interferes with the logic which combines characters into function-key codes. GPM also uses as part of its xterm control sequences a pair which save/restore the mouse mode (and are not actually handled by any of the other terminal emulators).

On writing this faq in 1999, there were no applications that used this misfeature. Later (in 2005) there were still no curses applications which do, however w3m contains some contorted code to exploit this, by abusing the library interface: it defines several symbols that conflict with ncurses to intercept calls to `wgetch`, while using other symbols from ncurses as is. (There is also documented `Gpm_Getch`, but it is no longer present in the GPM source code).

Since version 1.10 GPM comes with a configure script, which allows the system builder to suppress this from the shared library, e.g.,

```
configure --without-curses
```

You should verify that the shared library does not use the symbol `wgetch`. Version 1.16 lacked the configure script option to suppress this hook; removing libcurses.o from the list of objects in GPM's Makefile worked just as well. Version 1.17 built correctly when I tested it, however, though the changelog does not mention the change. It would seem that the issue would be long resolved. However, it is not.

Starting with ncurses 5.5, the recommendation is still the same: build the GPM library without the `Gpm_Wgetch` interface. ncurses 5.5 can dynamically load the GPM library on Linux, and that eliminates any reason to have the ncurses library built with an explicit dependency upon GPM.

Some of the GPM fixes are based on a quirk of the library: if `Gpm_Open` is called when the `TERM` environment variable contains "xterm", it opens a connection which returns the raw character stream which might contain mouse escape sequences. It returns a special file descriptor (-2) which is easy to overlook in the normal checks on file descriptors which are valid only when positive. For quite a while as well, GPM and X could not co-exist. It was not uncommon to have some safeguards to turn off GPM when starting X. By around 2005, some fixes had been made to the X server to allow the two to run concurrently. That exposed some problems in ncurses which had not properly checked for the special file descriptors, and affected programs running in an xterm, e.g., dialog and ded which start/stop the mouse interface. For these, the GPM server would ultimately be locked up and not return from a call to `Gpm_Open`.

Problems with Packages

Comments on packages

Packages are a good thing. Sometimes.

Not all distributions clearly distinguish between release versions of software, betas and alpha versions. (To be fair, not all producers distinguish these properly).

Ncurses 4/4.2/5.0

Ncurses 5.0 is not compatible with ncurses 4.2, however I frequently saw people advising others to "fix" programs that require the older library to make a symbolic link from the newer name to the older. That only works for simple applications, and not all of those.

Ncurses 4.0 and 4.2 were released respectively before and after X/Open finished their curses specification. Both were based on the draft specifications from 1995 and 1996. The released specification (available in 1997) differs in several places, mainly in the provisions for multi-byte character sets.

Late in 1998 through early 1999, I made corrections to the development version of ncurses to align it to the X/Open specification. Near the end of this, I realized that I had an opportunity to add an extension to ncurses which would make the terminfo format extensible, just as termcap is. It required a change to the `term.h` header. to allow the arrays for terminfo booleans, numbers and strings to be set at runtime. This had the effect of making programs not binary-compatible, but that was not a drawback, since it was already conceded that ncurses 5.0 would not be binary-compatible with ncurses 4.2 because of the X/Open changes. Only a few applications use `term.h`, and those would be fixed by a recompile.

One problem: Redhat packaged development versions of ncurses without distinguishing them from the release versions. We discussed the matter with them, but they did not wish to cooperate. Redhat 6.0 was released with almost all of the interface changes that comprised ncurses 5.0—as "ncurses 4.2". When ncurses 5.0 was released, they did not bother to read the release notes, and released that as "ncurses 4.2". Somewhat later, they added to the confusion by calling it "ncurses 4.0". Until mid-2001, much of this information was still available in Bugzilla.

Redhat continues to distribute development versions of ncurses without distinguishing between release- and development-versions.

rxvt's $COLORFGBG variable

Ncurses 5.2 added an experimental feature: support for rxvt's $COLORFGBG variable. This is a feature which tells the application what colors the default foreground and background correspond to. It is specific to rxvt: in general other terminal emulators assign colors for foreground and background which do not necessarily correspond to any of the ANSI colors. This feature was enabled in some rpm-based distributions, e.g., Mandrake and Redhat.

It worked for the configuration on which I tested, however there are two configurations. The format of the $COLORFGBG variable is not documented; you must read the C code to find how rxvt sets it. The two configurations correspond to whether the xpm library is used or not. If xpm is used, ncurses 5.2 sees the wrong value for the background, and display black-on-black.

Quick fix: unset $COLORFGBG.
Better fix: update the ncurses rpm (Mandrake did).

How do I report bugs?

First, check to see if your problem is addressed in this FAQ. Read the INSTALL document, if you have not done so. However, it may not be a known problem. Read on.

- How should I report bugs?
- How do I report problems building ncurses?
- Why aren't my bugs being fixed?
- How are patches organized?

How should I report bugs?

Contact the current maintainers at bug-ncurses@gnu.org.

To join the ncurses mailing list, please write email to `bug-ncurses-request@gnu.org` containing the line:

```
subscribe <name>@<host.domain>
```

This list is open to anyone interested in helping with the development and testing of this package.

There is an archive of the mailing list here:

http://lists.gnu.org/archive/html/bug-ncurses (also https)

Otherwise, you may email directly to the maintainers, currently:

- Thomas E. Dickey <dickey@invisible-island.net>

If you send email only to one of the other authors, I may not see it. We prefer that bug reports go to the mailing list. (Occasionally I get private email *cc*'ing Zeyd and Eric, neither of whom has contributed for more than ten years).

I get about half of my bug reports via the ncurses mailing list, some by reading news groups, and the others via direct email.

More than half of the changes that get introduced without review in the ncurses mailing list introduce a bug. So I find it necessary to review proposed changes.

When sending patches:

- Explain the problem that you are fixing.
- Please use "`diff -u`" format, even if you are creating a new file. Otherwise I may overlook it, if it is at the end of a long patch. (It is reasonable to send binary files in uuencoded form, but there are few of those).
- Don't send patches that include the '`configure`' script if the difference therein can be regenerated using `autoconf`.
- Do specify the version of ncurses that you are patching.

How do I report problems building ncurses?

This is a little different from reporting bugs. If you have a machine that I've not ported to, and have problems, I'll require the relevant information:

- config.log
- config.status
- include/ncurses_cfg.h
- log from running 'configure', with options
- log from running 'make', with options

A uuencoded/gzip'd/tar file is preferred, because the logfiles can be awkward to email. You may find the scripts which I use for building and saving logfiles useful.

If you're having trouble building on a known "good" platform, please make sure that you've got a current version of ncurses, and please read the installation instructions.

Why aren't my bugs being fixed?

Sorry. This is a hobby. There's a large backlog. Some changes pass review quickly, others are difficult, because one fix may break other functionality. My criteria are less stringent if you provide a short program that demonstrates the problem, or if you're modifying something that you maintain.

In any case, I will incorporate patches into my beta version only if I have reviewed the patch, tested it (if the patch is not obvious), and repaired any omissions (e.g., portability constraints). Occasionally I have patches (including my own) which cannot pass immediate review; these constitute most of my backlog. The remainder of my backlog consists of issues which highlight incompatibilities between ncurses and SVr4 curses; these are listed in the TO-DO file.

I use the following guidelines:

- extensions (deviations from SVr4 curses) are allowed only if they do not modify the documented/observed behavior of the API.
- extensions are feature changes which cannot be implemented without modifying the library.
- I test behavior using Solaris curses (2.5.1 or later), as well as SCO OpenServer and Tru64, and use the vendor's manual pages in conjunction with the X/Open Curses documentation.
- The Solaris XPG4 curses implementation is known to be badly broken; the SVr4 curses is usable for comparisons.

How are patches organized?

Prior to version 4.0 I posted patches to the ncurses mailing list summarizing only my changes (after applying changes submitted by others). The intent was that people who followed the list closely could build developmental versions.

Generally (unless we find a serious error), I issue patches on Saturdays, since validating patches takes time.

Beginning with version 4.0, I maintain "complete" patches (my changes together with those that I have integrated). It is simpler, and does not require making complete snapshots as often.

- User-supplied patches that require no modification are always tagged with their name, making it simpler to follow the revision history.
- User-supplied patches that require some modification are still tagged, but my patch will have followup changes.

Most files have RCS identifiers. If you are maintaining ncurses in an RCS (or CVS, etc.) archive, you can keep in sync with this using the "-k" option of `ci`.

Y2K Compliant?

Certainly.

The ncurses library does not store or retrieve dates in any form that depends on the year. Ncurses' use of time information is limited to

- computing the elapsed times, in fractions of a second,
- comparing file creation times, and
- sample programs that display the current time of day.

Terminology

Ncurses is used primarily with **terminal emulators**. A few people use it with hardware terminals.

Terminals

Short for *data terminals*, **terminals** provide users with the ability to enter and view data. The word *terminal* refers to something that is the endpoint.

Terminals are generally not part of the computer, nor are they necessarily near the computer. Originally, terminals were connected to the computer by cables, later by local networks. The terminal emulator running in your computer's desktop environment is a special case—still using a network connection, but perhaps sharing keyboard and display with other terminal emulators.

Emulators

An **emulator** of course is something which behaves like something else. In the case of terminals, these are programs which provide the same functionality as hardware terminals, or even other terminal emulator programs.

Control sequences

Often referred to as *escape sequences* because many begin with the ASCII escape-character, **control sequences** are sequences of characters sent to a terminal to make it perform some operation. Most terminals (the ones that ncurses deals with) are *asynchronous*, and accept control sequences which update parts of the display. There are others, e.g., the *synchronous* "block-oriented" terminals such as the IBM 3278.

The VT100's control sequences are used in many terminal emulators for example. In turn, most of those control sequences follow standards established by ANSI, ISO, ECMA. These are the most useful (and accessible):

- ECMA-35 - Character Code Structure and Extension Techniques
- ECMA-48 - Control Functions for Coded Character Sets

The standards do not give exact functional definitions. For that, ncurses development relies on vendor documentation, analysis and comparison testing.

Consoles

Originally **console** referred to the front panel of a computer, e.g., *lights and switches*, as well as the attached operator's terminal. The switches would do more than just turn the computer on and off.

- Some machines provided special purpose switches which could enter data or control the computer's operation.
- Later, the front panel was not so important for this purpose, and the operator's terminal was treated as the console. Many operating systems have a special terminal device for the system console terminal, e.g,.
  - `CON:` for MS-DOS
  - `/dev/console` on Unix-like systems.
  - `OPA0:` for VMS.
- Still later, with video terminals becoming more common than printing terminals, the console terminal became an integral part of the computer. Several operating systems implement their console terminal as part of the kernel, e.g., Solaris, FreeBSD, Linux. Many people are confused by this, and refer to these as *hardware consoles* though they are not. They are programs. Generally they are not referred to as *emulators* simply because they usually are not made to imitate another terminal.
- The console terminal now is used mainly when installing or reconfiguring the computer, rather than for routine control of its programs. Some system calls treat it specially, by using it as the destination for messages to the operator.

The X Window system can be run on many of the console terminals. Sun's workstations for example did this. Few people used the console terminal on that hardware (which by the way was not VT100-compatible). Since X took over the computer's display, some way was needed to make the operator messages visible. The xconsole program did that; desktop systems provide analogous solutions. It is not necessary to use a terminal emulator to display operator messages. A few terminal emulators (such as xterm) can intercept console messages as an optional feature. The konsole program on the other hand, does not perform this function, notwithstanding its name. Rather, it is a terminal emulator.

Microsoft Windows's treatment of console devices has changed:

- In MS-DOS, there was one display device.
- In Windows, there can be several console windows, each with its own console device. You can see the characteristics of the corresponding device by entering the command `mode con` into a console window. They are referred to as console windows because they provide partial compatibility with the MS-DOS console commands (including the device name `CON:`).
- They are not terminal emulators however: these console windows do not by themselves respond to *control sequences*. MS-DOS provided a special device driver for this purpose called `ANSI.SYS`.
- Windows provides an application programming interface (API) by which programs can treat the console window as a display.
- Several terminal emulators have been written using the Windows console API which run in one of these console windows (for example, the Cygwin terminal before mintty).
- There is also a port of ncurses using MinGW to these console windows. However, the ncurses MinGW port is a special case *not* using a terminal emulator. Like pdcurses, ncurses implements the Windows port using the Windows Console API. It does not rely upon `ANSI.SYS` or any of its successors.

Like the Windows Console API, neither ncurses or pdcurses are terminal emulators. However, it is possible to write a terminal emulator using any of these application programming interfaces.

Additional Reading

- For reference
- Language bindings
- Other implementations
- Related applications
- Technically obsolete, but often cited
- Interesting but misleading

For reference:

- The manpages, of course.
- X/Open Curses:
  - 19 Apr 2013 – Issue 7
  - 15 Jul 1996 – Issue 4, Version 2
- X/Open Curses, Issue 4, Version 2, Reference Pages
- Solaris 9 Reference Manual Collection,
  manual pages section 3: Curses Library Functions

- Solaris 9 Reference Manual Collection,
  manual pages section 3: X/Open Curses Library Functions
- AIX General Programming Concepts: Writing and Debugging Programs,
  Chapter 2: The Curses Library
- UnixWare 7 Documentation, Software Development,
  Character User Interface Programming
- NCURSES-Programming-HOWTO. (another copy here and source for the SGML and programs)

Language bindings:

- Ada95 binding, from ncurses
- CHARVA (Java binding)
- Common Lisp binding
- Eiffel binding
- JavaScript binding
- LinCRT (Kylix binding)
- OCaml binding
- Perl binding
- PHP binding
- Python binding
- Ruby binding
- Tcl/Tk Extensions & Information Page -> ncurses cTk

Other implementations:

- PDCurses
- NetBSD

Related applications:

- Xterminal's Home Page
- cdk Curses Development Kit (a library of widgets)
- dialog Script-driven curses widgets

Technically obsolete, but often cited:

- 4.4BSD Documents -> original curses
- Escribir Programas con NCURSES
- Writing Programs with NCURSES
- A Hacker's Guide to Ncurses Internals

Interesting but misleading:

- Text-Terminal-HOWTO.
  I find the deliberate misstatements in this one to be fascinating. Still, there is some useful information as well.
- nanocurses (Haskell binding)
  It says (referring to the ncurses manpages):

  > Sections of the quoted documentation are from the OpenBSD man pages, which are distributed under a BSD license.

- A C# binding for curses, which supports some of ncurses and PDCurses extensions, first noted in mid-2009 here (apparently added anonymously by the program's author). Its page claimed that "Virtually every function in curses has its managed counterpart." I noted that it implemented less than half of the functions in ncurses 5.7 (206/414). As of early 2013, it is (like most SourceForge projects), apparently dead.
- The treatment of escape sequences here is a mish-mash of incorrect information from the DOS/Windows and Unix environments. There are several areas of confusion, e.g., mistaking common practice for standardization (and introducing confusion about the scope of the standards), as well as over-generalization. The worst offenders are of course anonymous.
- The Python binding and tutorial bears some comment. On reflection, the comment is involved (see this page.

Copyright