UnixWorld Online: Tutorial: Article No. 006

Teaching Old Terminals New Tricks, Part 2 of 2

By Walter Alan Zintz.

Questions regarding this article should be directed to the author at walter@ccnet.com.

We saw how to deal with problems in a Termcap listing in the first installment of this tutorial series. Now we'll look at the analogous but more complex matter of Terminfo.

Strategic alterations to the Sun server's Termcap entry for our AT&T PC7300s solved several problems with most of the full-screen programs we use on the server. But they had no effect at all on flakey operation of Lynx--a World Wide Web browser for alphanumeric terminals--and checking everything in the Termcap entry that might have been causing this weirdness did not disclose anything that even looked suspicious. Sherlockian deduction: Lynx is not using Termcap.

[*Editor's note:* Lynx uses the curses library, which can take its information from either the Termcap or Terminfo database. Apparently, Lynx on the Sun server used by Walter's Internet service provider uses Terminfo not Termcap, as might be expected because `vi` uses a version of curses that employs Termcap, not Terminfo.]

Terminfo Enters the Picture

Could Lynx be using an internal, hardwired terminal handler? Possibly, especially because a system administrator once told me that Lynx only works correctly on VT100 terminals. Or could Lynx be using AT&T's answer to Termcap, Terminfo? An even more likely possibility. Tests to determine which is used here would be complex and time consuming; tinkering with the internals of the Lynx browser would be even worse; so the obvious route from here is to examine the Terminfo entry for the PC7300 to see whether the problem can be isolated there.

But Terminfo entries are not viewable ASCII files, because Terminfo is a compiled database. There is an original source file, of course, and quite likely a copy is laying around on the Sun server, but the current state of the compiled version may not correspond to the source file that happens to be online. So to save a lot of grief, I'll decompile the working Terminfo entry.

That's a job for a Terminfo program called `infocmp`. This oddly-named program does a bewildering variety of miscellaneous jobs, depending on the option flags used with it. In this case I'll use the `-L` option to decompile the PC7300 entry into comprehensible terms, not just cryptic codes. And because this program normally flashes its output onto the standard output device, usually the user's terminal screen, I'll have to divert it to a file. So my actual command line is:

`infocmp -L PC7300 > /tmp/verbose.pc7300.ti`

And the output in the `verbose.pc7300.ti` file looks like:

```
Terminal type PC7300
```

```
  7300|unixpc|pc7300|PC7300|unix_pc|AT&T UNIX PC Model 7300
flags
  auto_right_margin, eat_newline_glitch, xon_xoff,

numbers
  columns = 80, init_tabs = 8, lines = 24,

strings
  back_tab = '\E\t', bell = '^G', carriage_return = '\r',
  clear_screen = '\E[2J\E[H', clr_eol = '\E[0K',
  clr_eos = '\E[0J', cursor_address = '\E[%i%p1%d;%p2%dH',
  cursor_down = '\E[B', cursor_home = '\E[H',
  cursor_right = '\E[C', cursor_up = '\E[A', delete_line = '\E[M',
  enter_alt_charset_mode = '\E[11m', enter_bold_mode = '\E[7m',
  enter_dim_mode = '\E[2m', enter_reverse_mode = '\E[7m',
  enter_standout_mode = '\E[7m', enter_underline_mode = '\E[4m',
  exit_alt_charset_mode = '\E[10m',
  exit_attribute_mode = '\E[0;10m', exit_standout_mode = '\E[0m',
  exit_underline_mode = '\E[0m', init_1string = '^O',
  insert_line = '\E[L', key_backspace = '\b', key_beg = '\Ebg',
  key_btab = '\E\t', key_cancel = '\Ecn', key_clear = '\Ece',
  key_close = '\Ecl', key_command = '\Ecm', key_copy = '\Ecp',
  key_create = '\Ecr', key_dc = '\Edc', key_down = '\E[B',
  key_end = '\Een', key_eol = '\Eci', key_eos = '\Ece',
  key_exit = '\Eex', key_f1 = '\EOP', key_f2 = '\EOQ',
  key_f3 = '\EOR', key_f4 = '\EOS', key_f5 = '\E5',
  key_f6 = '\E6', key_f7 = '\E7', key_f8 = '\E8',
  key_find = '\Efi', key_help = '\Ehl', key_home = '\Ehm',
  key_ic = '\Eim', key_left = '\E[D', key_mark = '\Emk',
  key_move = '\Emv', key_next = '\Enx', key_npage = '\Epg',
  key_open = '\Eop', key_options = '\Eot', key_ppage = '\EPG',
  key_previous = '\Epv', key_print = '\Epr', key_redo = '\Ero',
  key_reference = '\Ere', key_refresh = '\Erf',
  key_replace = '\Erp', key_restart = '\Ers', key_right = '\E[C',
  key_save = '\Esv', key_sbeg = '\EBG', key_scancel = '\ECN',
  key_scopy = '\ECP', key_screate = '\ECR', key_sdc = '\EDC',
  key_sdl = '\EDL', key_select = '\Esl', key_send = '\EEN',
  key_seol = '\ECI', key_sf = '\Erd', key_sfind = '\EFI',
  key_shelp = '\EHL', key_shome = '\EHM', key_sic = '\ENJ',
  key_sleft = '\EBW', key_smove = '\EMV', key_snext = '\ENX',
  key_soptions = '\EOT', key_sprevious = '\EPV', key_sr = '\Eru',
  key_sredo = '\ERO', key_sreplace = '\ERP', key_sright = '\EFW',
  key_ssave = '\ESV', key_sundo = '\EUD', key_undo = '\Eud',
  key_up = '\E[A', newline = '\EE',
  parm_delete_line = '\E[%p1%dM', parm_down_cursor = '\E[%p1%dB',
  parm_insert_line = '\E[%p1%dL', parm_left_cursor = '\E[%p1%dD',
  parm_right_cursor = '\E[%p1%dC', parm_up_cursor = '\E[%p1%dA',
  scroll_forward = '\n', scroll_reverse = '\EM',
end of strings
```

A close look at these parameters shows one problem right off, in the sixth and seventh lines under the heading ‘‘strings’’. The command string to turn on bold characters and that to turn on reverse video are shown as identical. That would account for the worst problem when using the Lynx browser, so already I’ve confirmed that Lynx does seem to use Terminfo. I’m starting to unravel the Lynx problem, too, because a brief look back at my record of the attributes tests shows that ‘‘enter-bold-mode’’ is the one that’s incorrect, and that the only correction needed is to change the ‘‘7’’ to a ‘‘1’’ there.

Getting the Big Picture

Before I start making corrections, though, it would be a good idea to see whether there are any other errors in this Terminfo entry. The most likely way to turn them up would be to compare the items in this entry to those in my corrected Termcap entry, which seems to be working pretty well now. I could make an eyeball comparison, but that would take time and concentration, and I still might miss the mistakes that were the easiest for the original entry's writer to make--confusing the letter ‘‘l’’ with the number ‘‘1’’, for example.

Fortunately, I can have my computer do the comparison for me. The method may seem rather complex, but once it's understood it's straightforward. For starters my Termcap entry has to be in a file by itself--in this case I've named the file `pc7300.tc`. Next I'll run it through a program called `captoinfo`, which converts it to Terminfo source:

`captoinfo pc7300.tc > /tmp/ck.pc7300.ti`

This produces a few error messages, as `captoinfo` usually does:

```
captoinfo: TERM=s4: the string termcap code ’CV’ is not a valid name.
captoinfo: TERM=s4: the string termcap code ’CI’ is not a valid name.
captoinfo: TERM=s4: the string termcap code ’KM’ is not a valid name.
captoinfo: TERM=s4: cap ei (info rmir) is NULL: REMOVED
captoinfo: TERM=s4: cap im (info smir) is NULL: REMOVED
captoinfo: obsolete 2 character name ’s4’ removed.
        synonyms are: ’PC7300|unixpc|pc7300|3b1|Safari 4’
```

But none of these is a real error, and none is related to the sort of difference I'm looking for. The actual converted entry looks like this:

```
PC7300|unixpc|pc7300|3b1|Safari 4,
        am, xon,
        cols#80, lines#24,
        bel=^G, blink=\E[7m\E[2m, bold=\E[1m, clear=\E[2J\E[H,
        cr=\r, cub1=\b, cud1=\E[B, cuf1=\E[C,
        cup=\E[%i%p1%2.2d;%p2%2.2dH, cuu1=\E[A, dch1=\E[1P,
        dim=\E[2m, dl1=\E[1M, ed=\E[0J, el=\E[0K, home=\E[H,
        ich1=\E[1\@, il1=\E[1L, ind=\n, kbs=\b, kcub1=\E[D,
        kcud1=\E[B, kcuf1=\E[C, kcuu1=\E[A, kf1=\EOc,
        kf2=\EOd, kf3=\EOe, kf4=\EOf, kf5=\EOg, kf6=\EOh,
        kf7=\EOi, kf8=\EOj, rev=\E[7m, rmso=\E[m, rmul=\E[m,
        sgr0=\E[0m, smso=\E[7m\E[1m, smul=\E[4m,
```

If you think that this format is grossly different from the way the decompiled Terminfo entry looked above, you are so right. When I decompiled that entry I went for maximum readability, but normal Terminfo source, the only kind the Terminfo compiler `tic` understands, has a terseness similar to that of Termcap. To make a comparison with the source code generated from the Termcap entry, I'll have to decompile the Terminfo entry again, into the terser form, with:

`infocmp -I PC7300 > /tmp/pc7300.ti`

Next I'll convert each of the entries to be compared into a form that the Unix `comm` utility can handle, with each capability on a separate line. A fairly simple script for the `ex` editor will do this, and write the results to a new file called by the old filename with a plus sign (+) appended:

```
f %+
1d
```

```
%s/^ *
%s/, /,^m/g
%! sort
wq
```

Note that the `^m` in the fourth line is a control-M or newline. To get the `vi` editor to put that into the script, I had to type a control-V preceding it.

Making the Comparison

Finally, I use the Unix `comm` utility to find any differences between the two versions, as follows (from the `/tmp/` directory):

```
comm -3 ck.pc7300.ti+ pc7300.ti+ > comm.out
```

This produces a listing of just those lines that are in one file but not the other. In this case the lines that are only in the file generated from the Termcap entry will be flush against the left margin; those only in the file decompiled from the Terminfo entry will be indented. The actual output:

```
blink=\E[7m\E[2m,
bold=\E[1m,
        bold=\E[7m,
        cbt=\E\t,
cub1=\b,
        cub=\E[%p1%dD,
        cud=\E[%p1%dB,
        cuf=\E[%p1%dC,
cup=\E[%i%p1%2.2d;%p2%2.2dH,
        cup=\E[%i%p1%d;%p2%dH,
        cuu=\E[%p1%dA,
dch1=\E[1P,
dl1=\E[1M,
        dl1=\E[M,
        dl=\E[%p1%dM,
ich1=\E[1@,
il1=\E[1L,
        il1=\E[L,
        il=\E[%p1%dL,
        is1=^O,
        it#8,
        kBEG=\EBG,
        kCAN=\ECN,
        kCPY=\ECP,
        kCRT=\ECR,
        kDC=\EDC,
        kDL=\EDL,
        kEND=\EEN,
        kEOL=\ECI,
        kFND=\EFI,
        kHLP=\EHL,
        kHOM=\EHM,
        kIC=\ENJ,
        kLFT=\EBW,
        kMOV=\EMV,
        kNXT=\ENX,
        kOPT=\EOT,
        kPRV=\EPV,
        kRDO=\ERO,
```

```
	kRIT=\EFW,
	kRPL=\ERP,
	kSAV=\ESV,
	kUND=\EUD,
	kbeg=\Ebg,
	kcan=\Ecn,
	kcbt=\E\t,
	kclo=\Ecl,
	kclr=\Ece,
	kcmd=\Ecm,
	kcpy=\Ecp,
	kcrt=\Ecr,
	kdch1=\Edc,
	ked=\Ece,
	kel=\Eci,
	kend=\Een,
	kext=\Eex,
	kf1=\EOP,
kf1=\EOc,
	kf2=\EOQ,
kf2=\EOd,
	kf3=\EOR,
kf3=\EOe,
	kf4=\EOS,
kf4=\EOf,
	kf5=\E5,
kf5=\EOg,
	kf6=\E6,
kf6=\EOh,
	kf7=\E7,
kf7=\EOi,
	kf8=\E8,
kf8=\EOj,
	kfnd=\Efi,
	khlp=\Ehl,
	khome=\Ehm,
	kich1=\Eim,
	kind=\Erd,
	kmov=\Emv,
	kmrk=\Emk,
	knp=\Epg,
	knxt=\Enx,
	kopn=\Eop,
	kopt=\Eot,
	kpp=\EPG,
	kprt=\Epr,
	kprv=\Epv,
	krdo=\Ero,
	kref=\Ere,
	krfr=\Erf,
	kri=\Eru,
	krpl=\Erp,
	krst=\Ers,
	ksav=\Esv,
	kslt=\Esl,
	kund=\Eud,
	nel=\EE,
	ri=\EM,
	rmacs=\E[10m,
	rmso=\E[0m,
rmso=\E[m,
```

```
         rmul=\E[0m,
rmul=\E[m,
         sgr0=\E[0;10m,
sgr0=\E[0m,
         smacs=\E[11m,
         smso=\E[7m,
smso=\E[7m\E[1m,
         xenl,
```

At over a hundred lines, that's a pretty formidable list of discrepancies. I'm cheered up, though, by a brief inspection that shows many of these lines are not real problems. The very first line, the string to turn on blinking characters, is just my own attribute combination that I invented to replace the PC7300's lack of a blink attribute. Of course, it doesn't appear in the standard Terminfo entry. The next two lines tell me about something I've already discovered: the mistake in the Terminfo bold-characters item. All those items beginning with lower-case ‘‘k’’ that are in the standard Terminfo listing but not the one converted from Termcap? Most of them are for function keys that are defined in a separate keymap file that's incorporated by reference in the Termcap entry. The `captoinfo` program is not smart enough to follow this reference, and even if it were, the items in the keymap file do not use standard Termcap terminology. So half of the trouble has vanished under the light of knowledgeable inspection--although I should add those items from the keymap file to the basic Termcap entry, using standard terminology.

Dysfunctional Function Keys

It's less encouraging that all eight of the numbered function keys items, ‘‘kf1’’ through ‘‘kf8,’’ show gross discrepancies: the strings from the two entries don't even resemble each other. The strings for numbered function keys generally follow a series fron one key to another. The entry converted from Termcap shows this pattern, but the native Terminfo entry doesn't, so the latter is probably incorrect. It's easy to test the string that any key sends: enter the `vi` editor's insert mode and press the key in question, then see what's entered into the file. (All the alleged strings begin with the ASCII escape character, so to quote this control character into the file I must press control-V just before I press the key I'm checking.)

Testing two of the numbered function keys this way shows that in both cases the Termcap-derived entry is the one that's correct. A horrible thought seeps into my brain: perhaps the native Terminfo descriptions of the myriad other function keys' strings are also incorrect? Checking a randomly-chosen five of these keys shows that, in every case, the native Terminfo description bears no resemblance to what the key actually sends. This is perplexing enough that I have to investigate the whole matter before I start making changes.

How can Lynx run at all with garbled key-send items? Well, none of us uses any function keys except the four cursor arrows when running Lynx. A quick check as above showed that the cursor-arrow descriptions are not defective, which is why the problem did not show up earlier. Second, while a few typographical errors are par for the course when a new entry is written, how could this much utter nonsense appear in a single entry? And why didn't any of the myriad Sun and PC7300 users out there discover and correct this long ago?

Those latter questions may never be answered. My first thought was that the key-send items may have come from someone running the `captoinfo` program on the wrong Termcap listing. This sounded just probable enough that I picked out four of the known defective items, translated them to Termcap format by hand, and searched the `/etc/termcap` file to see whether they all came from another terminal's entry. No entry had more than one of these key-send descriptions, and two of the four descriptions were not in any of the entries. Could the strings simply be associated with other keys, so that only the key-name to string-sent links were wrong? An eyeball check showed no sign of this. Most perplexing of all, the strings that native Terminfo alleged were sent by these keys were usually mnemonics for the key names--the ‘Move’ key’s alleged string was `\Emv`, the ‘Copy’ key’s was `\Ecp`, and so on. Because these key-strings are strictly for keyboard to CPU communications, and machines have no use for mnemonics, this aspect is utterly unexplainable.

But this problem was as easy to fix as it is difficult to explain. I did not need to know exactly which key-send items were wrong; didn’t even need to know the correct strings! I just went into the Terminfo source file with the `vi` editor, went to the start of each incorrect or merely doubtful key-send string, typed a ‘‘change’’ command, and typed as the new string a control-V followed by pressing the function key whose string I wanted to insert. Pressing the function key sent the actual key-send string to `vi`, just as if I’d typed it character by character, and the control-V quoted in the escape character that begins every string sent by a function key. Finally, I did a global substitution to replace each escape character with the ‘‘\E’’ string that represents it in a Terminfo or Termcap listing, and the job was done.

When the Going Gets Tough, the Tough Get Going

And that was the end of the easy improvements. From this point on it was all grinding detail work, and without any manuals for our PC7300s I had to answer every question the hard way. Sometimes that meant creating a file containing a string I needed to test, then using the Unix `cat` command to send that string to the terminal. More often I could use Terminfo itself to simplify the testing. I had three Terminfo entries in source form: the native one, the one I had converted from Termcap, and a third that a helpful user had sent me, containing his and his friends’ fixes for these problems. On any item where these differed, and that included most items, I could run Lynx alternating between the three Terminfo entries to see which worked the best. Of course, that meant I first had to set up a way to switch Lynx between the three Terminfo entries.

(At this point it’s reasonable to ask why I should bother with all this work when I have a Terminfo version of the Termcap entry that has been working pretty well. It’s tempting to stick with that version, but there are reasons against it:

- There are many things that `captoinfo` can’t convert, or can’t convert correctly. That keymap file is a good example.
- `captoinfo` sometimes adds information that’s not in the Termcap file, basically by guessing. For instance, if the Termcap entry doesn’t specify a string to sound the terminal bell, `captoinfo` will assume that the ASCII `bel` character, control-G, is what does it.
- Quite a few capabilities and limitations that can be expressed in Terminfo just have no Termcap equivalents.
- There may well be flaws in my reworked Termcap entry that haven’t shown up in use so far, as the erroneous function key send-strings in the native Terminfo entry didn’t show up because none of us happened to use those function keys with our one known Terminfo-dependent program, Lynx.

All in all, a little more work with the Terminfo entry seems likely to save future trouble, and may fix some current problems we just didn’t know we had.)

As with Termcap, there’s a standard place for Terminfo-driven programs to look for the compiled data on terminals, and as with Termcap this can be overridden by an environmental variable. With Terminfo each terminal's description lives in its own file, so the environmental variable must be set to the full pathname of the top directory above the individual files. The native Terminfo entry was already set up in the standard place, so my first task was to use the `mkdir` command to set up directories for the from-Termcap and from-a-benefactor versions of Terminfo. I called them `/u3/walter/terminfo2/` and `/u3/walter/terminfo3/`, respectively. (Because I expected to be working on the Terminfo corrections off and on for some days, I did not want to put these entries in such a volatile place as `/tmp/`.)

The Terminfo compiler--`tic`--also takes care of placing the compiled entries in the proper place. It checks for an environmental variable that gives a special place for programs to seek compiled Terminfo entries, and if it finds one it puts the compiled output in that place. If none is found, it puts its output in the standard Terminfo location, ordinarily subdirectories of `/usr/lib/terminfo/`. Because I need to put the Terminfo entry I've derived from termcap into `/u3/walter/terminfo2/`, I need to set that environmental variable to point there before I compile the source code into a working entry--that will automatically put the output where I want it, as well as keeping this variant entry from overwriting the native entry. I set this variable the same way I set the `TERMCAP` environment variable in the first part of this tutorial. Because I'm using the C shell on this system, I type:

```
setenv TERMINFO /u3/walter/terminfo2
```

Now all I need to finish the task is to type:

```
tic /tmp/ck.pc7300.ti
```

Compiling the donated entry follows the same pattern.

Now I can use either of the new Terminfo entries, simply by setting the `TERMINFO` variable to the correct value, and I can go back to using the native entry by unsetting that variable. So now I'm ready to run practical tests on any Terminfo items.

A Comprehensive Example

One absolutely fundamental terminal capability is being able to send the cursor from wherever it is now to a given row and a given column. In Terminfo this is denoted by the `cup=` item; it's classified as a string item, but actually it's a way to generate a string that carries the correct information for the particular row and column the program wants to move to. My three Terminfo entries have different formulas for this item:

native Terminfo:
```
\E[%i%p1%d;%p2%dH
```
from Termcap:
```
\E[%i%p1%2.2d;%p2%2.2dH
```
donated entry:
```
\E[%i%p1%2.2d;%p2%3.3dH$<6*/>
```

Before I started analyzing the code, I checked to see how well each of these items worked in practice. To do this I set the `TERMINFO` variable to each of the three locations in turn, each time pointing Lynx to a Web site where the buttons are scattered around the screen and seeing how well the cursor moved when I jumped from one button to another. Both the native Terminfo entry and the one I had converted from

Termcap seemed to work exactly as they should--I couldn’t tell them apart just by watching the cursor make its moves. With the donated Terminfo entry, cursor placement was always accurate, but motion was very slow and rather jerky.

Now down to analysis. The three items agree that the string to be sent to the terminal for an absolute cursor move begins with the ASCII escape character followed by a left square bracket (`[`), has a semicolon (`;`) in the middle, and ends with a capital ‘‘H’’. Nothing else in any of the items is sent verbatim to the terminal; rather, it all tells the program how to calculate what should be sent to indicate the row and column on which the cursor should land.

I’ll start with the string `$<6*/>` that appears after the capital ‘‘H’’ in the item from the donated entry. A dollar sign followed by a number within angle brackets is the Terminfo signal that the host computer should insert that many milliseconds of padding at that point. The asterisk (`*`) says that if the move involves a jump of more than one line, there should be that many milliseconds of padding for each line moved. The slash mark (`/`) indicates that this padding is necessary even when terminal communication is via the XON/XOFF protocol. (This last is an example of a Terminfo exclusive that has no equivalent within Termcap.)

A check of the source listing for this entry shows that quite a few other items have associated padding, including items such as turning certain character attributes off and on. All this padding easily accounts for the slow, stuttering progress when moving from button to button within Lynx while running at slow line speeds (we’re using the PC7300 internal modems, which max out at 1200 baud). There is another item in this entry to tell the program that padding is not needed when running at less than 9600 baud, but not all systems think this applies generally, and Lynx appears to be one of those exceptions. In sum, this padding is admittedly not needed when running only 1200-baud modems, and its appearance in the entry is causing trouble, so out it goes.

Getting back to the main sections of the three items, the percent sign (`%`) that appears so frequently in them is the signal that what immediately follows is an instruction. (To put an actual percent sign in a Terminfo or Termcap item, type it twice.) Terminfo is equipped with a virtual calculator: it has a stack, uses reverse Polish notation, and has arithmetic, boolean and bitwise operations. These percent-sign notations all serve as instructions to the virtual calculator.

The three items all have a `%i`, which means that the PC7300 considers the leftmost column and the topmost row as being numbered one rather than zero. In each item the semicolon separates the method of indicating the row (on the left) from the method of indicating the column. And the `%p`*n* tells the program to push the *n*th parameter onto the stack, while the `%d` tells it to pop the most recent value from the stack, as a signed decimal number, and put it in the string at this point. (Does that seem like an awfully longwinded way to say that to move the cursor to row 7 and column 23 you simply send `(escape)[7;23H` down the line? It is, but most terminals use much more complex encoding.)

The only point on which the items disagree is how those numerical values are to be formatted. The characters between a percent sign and the following ‘‘d’’ are formatting instructions, which generally use the `printf()` notation. So the native Terminfo item calls for no special formatting, the item converted from Termcap says that both output numbers must be padded to two digits, using leading zeroes if necessary, and the donated item asks for two digits in the row number but three in the column number.

But as I discovered in my preliminary tests, all three versions put the cursor in the correct row and column. So do I go with the native Terminfo item for the sake of simplicity, or with the donated item on the grounds that circumstances may crop up some day that will require this formatting? (The from-Termcap item is not likely to be correct because the PC7300 has modes in which there are more than 99 columns.) This is where personal judgement comes into play. Whatever the decision, it's a very good idea to go back and change the working Termcap entry to be the same, just for consistency.

The rest of the debugging work is pretty much more of the same. Most of it won't be as complex as the example we've just gone through, because absolute cursor movement is one of the most complicated functions a terminal performs. But all the work will involve comparing items to each other, analyzing them with the help of a Terminfo reference, and testing at the start and (often) at various points thereafter.

Investing Your Intellectual Capital

There's a lot of work to be done here, but I can get more out of it all than just correct Termcap and Terminfo entries. Here are a few examples of uses for terminal-operation data that do not involve Termcap or Terminfo:

- Occasionally a program crash or other calamity will leave my terminal set in some odd modes: reverse video, cursor invisible, etcetera. Now I'll know how to write a file that I can `cat` whenever this happens that will reset all the terminal modes to normal.
- At times I tend to forget whether I'm connected to the Sun server or working directly with my PC7300. To vividly help me remember, I can go into my `.login` file on the Sun server and modify my prompt visually. At the start of my prompt string I'll put the characters that turn on, say, the combination of bold and dim characters; very distinctive visually. At the end of my prompt string I'll put the characters that turn off fancy video modes. Then my prompt on the Sun server will appear in unmistakeable form, while everything else appears normally.
- Shell scripts that will be used exclusively with our PC7300 terminals can be fancied up with character attributes and such in the prompts and in the output, using the same approach as in the list item above.

The only special point to remember is that, because these uses do not involve Termcap or Terminfo, the metacharacters they use will not work here. That is, instead of using the `\E` sequence to represent the ASCII escape character, it will be necessary to put the escape character itself in the string.

Edited by Becca Thomas / Online Editor / UnixWorld Online / beccat@wcmh.com

Go To Content   Search

Last Modified: Thursday, 07-Sep-95 12:55:39 PDT