UnixWorld Online: Tutorial: Article No. 002

Teaching Old Terminals New Tricks, Part 1 of 2

By Walter Alan Zintz

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

Can alphanumeric terminals and desktop PCs from the 1980s and early 1990s handle the features of today's full-screen applications? Not of Mosaic and the like, of course; just modern full-screen alphanumeric interfaces that come with programs like the Lynx Web browser and various front ends to Unix. Experience seems to say ‘‘No; we've tried using them and they can't keep up’’. But most often the limitations are not in the terminal or the terminal section of the desktop system; they're in poorly-done configuration files and databases, especially Termcap and Terminfo entries.

Necessary Hazards

Termcap and the later Terminfo started out as a great concept, even a necessity. When display terminals moved beyond the ‘‘glass TTY’’ stage, more control of the screen was needed than could be provided by the 32 ASCII control characters, and every new terminal seemed to have a different set of escape control sequences for the additional functions. To let a full-screen program work with more than one or a few models of terminals, it was necessary to create a database of different terminals' idiosyncratic control sequences, so a program could look up what it needed to know about whatever terminal it was connected to. But these databases have been shakey from the start.

Back then it was often the case that, while starry-eyed hardware engineers were building terminals with capabilities galore, the software department was more pessimistic. Someone in software would say, "Our customers don't need half those features --don't bother to write Termcap items for the stuff they won't use because it would only get them in trouble". That's if we were lucky--otherwise the terminal manufacturer wasn't concerned with Unix and released his product with no Termcap entry at all. Then one of the first Unix-using buyers would cobble up a bare-bones Termcap entry to meet personal needs, start passing it around among friends, eventually post it on the net--and so a miserably inadequate but quasi-official standard was born.

When a terminal's performance is inadequate or erratic on full-screen programs, a little detective work starting with Termcap or Terminfo will very often set things right, and having no documentation for the terminal is usually not a fatal obstacle.

Sherlock Takes the Case

A good example of all this was my recent effort to link some vintage desktop systems to the Internet.

The desktop systems were AT&T PC7300s, with small monochrome screens and little more than alphanumeric capabilities. They had been purchased at a closeout sale, without manuals, and AT&T said it had no PC7300 manuals to sell. Nonetheless, with intuition and knowledge of Berkeley Unix we'd been able to do everything we wanted to with them, up to the present.

But now I wanted to connect them to the Net, via shell accounts at an access provider running Sun servers. At first we couldn't even log into the system. As soon as the menu front end came up, it printed the start of an error message and disconnected the line. A system administrator there finally gave me direct access to command-line Unix, after which I could log in, at least. Both PC7300s and Sun servers are based on Berkeley Unix, which uses Termcap, but a look around showed that these Suns had no entry for the PC7300 in their Termcap files. This was a surprise--most vendors use the Berkeley-standard Termcap file for all but their own terminals, and the PC7300 has been in the standard file for years. I uploaded the Termcap entry the PC7300s use internally, the system administrator installed it, and then we could get into full-screen programs, including that menu front end.

But running those full-screen programs generated problems. There was some strange behavior, mostly involving character attributes such as reverse video and underlining. Ominously, this strangeness was not consistent: various surprises cropped up while running different programs.

The worst problem was with Lynx, a Web browser designed for alphanumeric terminals. A Web document may have several phrases the user can click on within a single page. Because Lynx is intended for systems without mouse pointers, it is supposed to use one attribute mode to display the phrase that would be activated by pressing the return key now, and another attribute to show the phrases the user could move to with the up and down arrow keys. With our PC7300s, unfortunately, Lynx usually displayed all the phrases in the same mode: reverse video.

Discouraging? Actually, the problem was halfway to being diagnosed when I thought over that symptom list. When character attributes malfunction differently in different programs, the problem is almost certain to be missing information in the Termcap or Terminfo entry. Programs that fail to find an entry telling how to produce one character attribute mode must default to another mode or skip the attribute altogether. Each programmer makes his own decision about these substitutions, so each program defaults in its own way. The Lynx problem just as certainly indicated that character attribute information was present but not correct. No sane program replaces a missing attribute with one it already uses for another purpose on the same screen.

Scrutinizing the Evidence

The first thing to do was make a working copy of the PC7300 Termcap entry, reproduced below:

```
s4|PC7300|unixpc|pc7300|3b1|Safari 4:\
        :md=\E[1m:me=\E[0m:\
        :al=\E[1L:am:bs:\
        :cd=\E[0J:ce=\E[0K:cl=\E[2J\E[H:cm=\E[%i%2;%2H:co#80:\
        :dc=\E[1P:dl=\E[1M:do=\E[B:ei=:ho=\E[H:\
        :ic=\E[1\@:im=:kb=\10:kd=\E[B:kl=\E[D:kr=\E[C:ku=\E[A:li#24:\
        :k1=\EOc:k2=\EOd:k3=\EOe:k4=\EOf:k5=\EOg:k6=\EOh:k7=\EOi:k8=\EOj:\
        :nd=\E[C:se=\E[m:so=\E[7m:ue=\E[m:up=\E[A:us=\E[4m:\
        :EE=\E[m:BO=\E[1m:DS=\E[2m:XS=\E[9m:CV=\E[=C:CI=\E[=1C:\
        :KM=/usr/lib/ua/kmap.s4:
```

That's not quite as cryptic as it looks. The items between the vertical bar symbols (`|`) are alternative names for the PC7300 (and for related machines that use exactly the same codes). Between the colons you'll find terse descriptions of what the PC7300 can do and the character strings needed to drive it.

A pound sign (`#`) in the middle of an item indicates a numeric value, as in `co#80`, which tells the calling program that the screen holds 80 columns.

An equal sign shows the character string needed to produce an effect. For instance, in `cd=\E[0J`, the `cd=` directive indicates that what follows will, when sent to the terminal, clear everything the screen from the cursor onward. A `\E` represents the ASCII escape character (a control-[), and a backslash (`\`) followed by two or three digits represents in octal the value of the ASCII character to be placed there.

An item with neither a `#` or an `=` within it states that the terminal has a certain capability. For instance, `am` says that when a line of characters is about to overflow the right edge of the screen the terminal will wrap it around to the next line.

Comparing lines 2 through 8 of the Termcap entry above to my Termcap reference showed that there were items giving the strings needed to turn on underlining (`us=\E[4m`), extra-bright characters (`md=\E[1m`), and whatever character attribute the configurer regarded as particularly distinctive, called standout mode (`so=\E[7m`); but no other character attributes. Whatever this Termcap entry's standout mode might be, this collection of attributes is not enough for all full-screen programs, which could explain why some of them were not painting screens correctly.

The last two lines contained items from a Termcap extension called Tam, used almost exclusively on AT&T systems for drawing windows and forms. One item gave again the string that starts extra-bright characters; two others enabled dim and overstruck characters, respectively. But most programs cannot read Tam capabilities.

The next question was whether the additional character attributes were missing from the PC7300's capabilities, or only from the Termcap entry. I had no PC7300 manual to consult, but I did notice that the attribute-on strings I'd found in the Termcap entry were identical except for one numeric digit in each string. Might it be that substituting other digits would bring out other attributes?

To test this idea I wrote a single-line file that read `abc^[[9mdefg^[[0mhij` in full. By sending contents of this file to the screen (with `cat`), I would get the first ten letters of the alphabet displayed, with the letters "defg" in whatever attribute style the numeral 9 produces, if any. (Note that since `cat` does not use Termcap, I replaced the `\E` with the actual ASCII escape character, control-[; and that I followed "defg" with the string said to turn attributes off again, so I wouldn't leave my terminal locked in an attribute mode.) I tested all ten numerals this way, with mixed results. No new attributes turned up, but all the strings Termcap and/or Tam gave as attribute switches worked as promised (standout mode turned out to be just reverse video).

One of the items in the Tam section, `EE=`, which gives the string that turns all attributes off at once, also implies that any or all of the character attributes may be combined to yield still other attribute modes. I tested this implication by putting multiple attribute-on strings in my file to be displayed and the combinations all functioned. Even the unlikely cases proved workable: all five attributes at once produced a distinct though unattractive effect; and combining extra-bright with dim, which too many programmers assume is a no-op, created one of the nicest typefaces of all.

Taking Action

Next I had to make use of all this new information. I started by writing a revised Termcap entry in which every attribute I'd found got an item in both the standard and Tam vocabularies. (As far as this was possible; there is no standard Termcap item for the overstruck attribute, for instance.) Not that we could expect to encounter Tam-using programs on a Sun server, but this would let us have identical Termcap entries on the Sun servers and our own PC7300s, which keeps things simple.

Putting new items into the existing entry only required inserting each of them in one of the lines, with a colon before and after each new item. The order of the items in a Termcap entry is not significant in operation, and human readability was already hopeless in this entry. To keep the new Tam items with the existing ones I created a new line, which required putting 8 spaces at the start of the line and escaping the ending newline with a backslash.

That left the problem of blinking mode. Some programs use blinking text, and the PC7300 has no way to blink anything but the cursor. In the end I resolved this by setting the Termcap blink item to a string which turns on a combination of reverse video and dim--this produces reverse video with a screened background. It doesn't blink, of course, but it's visibly different from any of the other character attributes. To remind users that this mode is what they should look for when a program displays messages about ‘‘the blinking text’’, I wrote a text file that held just the single line ‘‘Text in this typeface should be regarded as blinking’’--with the words ‘‘this typeface’’ surrounded by the codes that would display them in my pseudo blink mode--and I put a line in everyone's C shell initialization file (`.login`) that would display this new file at login.

With reverse video declared as a mode of its own within my revised Termcap entry, having standout mode also be reverse video was redundant. Someday we might come across a program that used both those attributes, and there would be no way for users to tell the standout-highlighted characters from the ones using reverse video under its own identity. So I revised the entry again, to make reverse video combined with extra-bright the new standout mode.

Although I had less to go on in tracing the other minor problems--most often a few stray characters here and there--I decided to make a cursory check of the Termcap entry for possible causes. The entry's assertions about what the PC7300 does-- automatically wraps overlength lines, backspaces, has 80 columns and 24 lines--I knew to be correct. I knew nothing about the strings to move the cursor, erase all or part of the screen, etcetera, but those are easy to test. I only needed to put the string alleged to perform a task into a file (using actual escape characters instead of Termcap representations), then send the contents of the file to the screen (with `cat`) to see that the action was as expected. (In a few cases I would have to use more than one such string in a file to run a test accurately. For instance, the string to erase from the cursor to the end of the screen. When working from the command line, the part of the screen following the cursor is normally empty, so it's necessary to precede the erase-to-end string with another string that moves the cursor up into the already-written-on screen area in order to see something get erased.) I tested a few of the likely-suspect strings this way, but found nothing incorrect.

Putting it to the Test

The last question was how to put my revised Termcap entry into effect on the Sun servers. The `/etc/termcap` file generally can't be modified without the superuser password. System administrators usually don't want to put an experimental entry like this into the main Termcap file, nor did I want to have to hunt up a system administrator every time I needed to make another little Termcap change. But there are other places where a Termcap entry can live. It's possible to set a `TERMCAP` variable within a user's environment to override the default selection of `/etc/termcap` as the place where programs look for a Termcap entry. This variable can be set from the command line, or set automatically at each login by putting the appropriate definition line in the user's shell initialization file (`.login` for C shell and `.profile` for Bourne and Korn shells).

If the `TERMCAP` variable is set to a string that does not start with a `/` mark, then the string itself is taken as the Termcap entry. A possibility, but rather cumbersome when the entry is subject to continuing experiments and changes. But if the variable string does begin with a `/` mark, it's taken as the full path name of a file to be used in place of the `/etc/termcap` file. That's the convenient place for a Termcap entry that's likely to be tested and modified from time to time, which is why I chose it.

What I'd done so far should have cleared up most of the character attribute problems. Testing showed that it did--Hallelujah! No new problems had cropped up along with the improvements, so I set all our users' `.login` files to use my revised Termcap entry, and downloaded it to our PC7300s so we could try it there, too.

But I'd found nothing that could be causing the Lynx attribute confusion or the other miscellaneous problems not involving attributes, so I wasn't disappointed when I didn't see any improvement there. Next time I'll explain how I cracked those problems.

Termcap Documentation

If you think you just need the `man` pages to fathom Termcap, but they're not online on the system you're using right now, the place to find them is the *4.4BSD Berkeley Software Distribution* book series, especially the *Programmer's Reverence Manual* volume. These are published jointly by the Usenix Association and O'Reilly & Associates, and should be around any site that uses Berkeley Unix.

But if you are about to buy Termcap documentation, consider instead *termcap & terminfo*, a book by John Strang, Linda Mui and Tim O'Reilly that is published by O'Reilly & Associates. This book explains both the systems named in its title without the terseness of `man` pages. A third of the pages overview the two systems and offer strategies for using them. Most of the rest is discussion of individual capabilities, grouped into chapters with related items. The finale is an alphabetical guide to capability descriptors, cross-referenced between Termcap and Terminfo. (I only wish the references to text pages were more direct.) The entire book is pretty lucid and pretty accurate. It's paperbound, has 253 pages, and is priced at $21.95.

My other caveat regarding these books is that they don't even mention *ad hoc* Termcap extensions such as Tam. You may not need to know about Tam, but if you do you'll have to find the online `man` pages for it, or pay close attention to the summary of Tam capability descriptors attached to this tutorial.

You can order O'Reilly books directly from the publisher at:

- Web: http://gnn.com/ora/
- E-mail: order@ora.com
- Toll-free: (800) 889-8969

(*Return to main tutorial*)

Describing Tam Capabilities

The Tam capability items live inside the standard Termcap entry, usually on a few lines by themselves at the end of the entry. But they're used only by programs that understand the `tam(3T)` system, which is primarily for drawing windows and forms on alphanumeric terminal screens.

There are just a dozen Tam items for termcap entries, and all of them are string variables. They're expressed in standard Termcap style: first a two-letter code to indicate what capability is being given, then an equal = sign, finally the string that the program should send to the terminal to perform the action.

Most of the items deal with character attributes. Tam uses `BO=` to turn on extra-bright characters, `DS=` for dim characters and `XS=` for overstruck characters. The corresponding items to turn these attributes off are `BE=`, `DE=` and `XE=`, respectively. But you may not need those last three items. If there is one string that will turn off any and all attributes, and if (very important) any or all character attributes can be combined on this terminal without restriction, then use `EE=`to give the string that turns off everything and skip the individual attribute-off items.

Tam's use of these attribute items interacts in complex ways with the standard Termcap items `so=`, `us=`, `sg#` and `ug#`. Be sure that any of these latter which have values are set correctly.

An item `CI=` gives the string that turns the cursor invisible on the screen; `CV=` tells how to make it visible again. If the terminal you're describing has hardware SLK keys, the string following `FL=` should put a label on a key and `FE=` should turn off the SLK labels. The string with the `FL=` item must be a `printf` format string with provision for two runtime arguments: the number of the key to be labeled and the text which will be the label.

Finally, the `KM=` item gives the full path name of a file which tells what strings the various function keys send. Customarily, such files are in the directory `/usr/lib/ua/`, and have names that start with `kmap.` and end with the usual short name for the terminal being described.

(*Return to main tutorial*)

Letter to the Author

June 10, 1995

Dear Mr. Zintz:

I saw and enjoyed your article ‘‘Teaching Old Terminals New Tricks’’ in UnixWorld Online, but as a former AT&T UNIXpc enthusiast (I still have several of the poor dead things lying around my house), I feel I must object to the characterization:

> ‘‘The desktop systems were AT&T PC7300s, with small monochrome screens and little more than alphanumeric capabilities.’’

Well, to be honest, I can’t argue with the ‘‘small monochrome screens’’, but I will take issue with the latter part. The UNIXpc had full bitmap graphics, a mouse, and a desktop windowing environment with the stock OS. Also, MGR, the Bellcore window manager--now enjoying a small resurgence in popularity with some Linux users--was available and widely used. Actually, I prefer MGR to the X11 desktop I’m using right now; unfortunately, MGR never really took off, and so lacks clients, and other support that have made X popular.

Anyway, while the UNIXpc is now well past its prime, it was at one time fairly advanced, quite useful, and certainly had more than ‘‘little more than alphanumeric capabilities.’’

John R. MacMillan / Toronto, Ontario, CANADA

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

Go To Content Search

Last Modified: Wednesday, 30-Aug-95 08:44:52 PDT