How I use WM

Tom Truscott

Research Triangle Institute
P.O. Box 12194
RTP, NC 27709
(919) 541-7005

Starting up wm

The last line in my .profile is

```
exec wm
```

which puts me into the window manager, so I am always in it. When I need a real terminal (e.g. to run 'talk') I can type <*wm-esc*>z. Since I go right into wm it is important that TERM be set correctly. Earlier in my .profile I have

```
export TERM
case "$TERM" in
sw|su)      TERM=‘whatterm‘;;
esac
```

So that if the TERM variable is su or sw (dataSWitch) John Menges' whatterm program is invoked to figure things out. (Whatterm depends on the response characteristics of a dozen or so terminals. A more general version, using a 'responsecap' database, is planned.) Whatterm is doomed to fail on an ADM terminal (unless it has the HERE-IS option) and also fails if the user is already in wm and and logging in to another computer, because 'wm' itself does not respond to any escape sequences. It probably should, so whatterm will work! Actually, nested invocations of wm, while amusing, are probably unnecessarily confusing.

Choosing window layout

That is largely a matter of individual taste. My ''#1'' window is full-screen (except for the bottom line), my #2 window is the top half of the screen and my #3 window is the bottom half (with an unused line between them), and my #4 window is just like #1. I am normally (99%) in the #1 window and am generally unaware of the existence of wm. But when I want to escape from the current window I can change to window #4 to get a fresh screen (one wm nuisance is that it puts me back in my home directory on my home machine). Or if I want split screens, say for debugging or file comparison, I can change to window #2 and then alternate between #2 and #3. I think of having three window layers, the #1 layer, the #2/#3 layer, and the #4 layer. (In reality, wm maintains a window display order, and changing to a window puts that window on 'top', so there is not really a #2/#3 layer.) On a 66 line display I bet things would be much nicer. Some people have a window which covers all but the top line, so they can put a clock or periodic 'uptime' in the top line. (Note: two windows *must* have an unused line between them or else they cannot be used for simultaneous displays.) Also, some people change the wm prefix character to '^W' or '^P' so that 'vi' is easier to use.

The undocumented experimental <*wm-esc*>f ('fit') command is also handy. For example, suppose I am in my #1 window and type ''make''. It might take a while, so I would like to switch to another window and read news. But I would also like to see the progress of the make. So I switch to a half-screen window and type '<*wm-esc*>f 1', which relocates the #1 window so that it is no longer obscured. Now I can read news and keep up with the ''make'' at the same time. This can really jumble up the windows, but in this case I can change back to #1 and do a '<*wm-esc*>f 1' to make it full-screen again.

Window ‘‘Performance’’ Considerations

There are some important performance details to consider. First, your terminal needs insert-line/delete-line or scrolling region support for wm to work well on anything but full-screen windows. In particular, my #2/#3 windows are sloowww without such support. The scrolling region is best because with insert/delete-line scrolling in the #2 window causes the #3 window to jitter up and down. By the way, the scrolling region capability is often omitted in many termcap entries, so you might need to put it in. Here is the relevant vt100 stuff, which is probably the only scrolling region support you are likely to find:

:sc=\E7:rc=\E8:cs=\E[%i%d;%dr:sr=5\EM:

For now, all of ‘‘cs’’, ‘‘sc’’, ‘‘rc’’, and ‘‘sr’’ must be specified or wm will not use scrolling regions.

By all means you should make windows be the full width of the screen. Without that wm scrolls by redrawing the whole window. The painful slowness is tolerable for smaller windows, and also for faster terminal speeds. My office terminal is at 4800 baud, and wm is okay at that speed, but even 19200 is probably too slow for a window that is full-screen except for the right-most column.

WM Enhancements

The ‘‘Hacking WM’’ document mentions several possible enhancements. Suggestions for other improvements to wm are welcome. Code for any new improvements are even more welcome!

Hacking WM
(Obscure Details for the Hard-Core User)

Tom Truscott

Research Triangle Institute
P.O. Box 12194
RTP, NC 27709
(919) 541-7005

In Praise of Matt Lennon

Matt Lennon took an interest in Robert Jacob's Usenet wm distribution, got it running here, and talked others into trying it out. Frankly, the original version was too slow to be usable, although it was indeed interesting. (Perhaps most interesting of all was that Jacob had a version of wm running under UNIX V6!) Matt reorganized the program internally and put in the first of a series of scrolling hacks (in WMinsertln and WMdeleteln). The resulting program shares with the original wm a simplicity of design that is important in any program, but particularly in a window manager. I think of the revised wm as the ''A'' version. I am afraid a ''B'' version will be along all too soon.

Problems with Running Programs under WM

There are three common problems when running programs under wm. One problem is that some programs assume the screen is at least some minimum size, such as 24 lines and 80 columns. Typical programs betray themselves by scribbling text intended for ''line 24'' at random places on the screen, or by dumping core after the ''newwin'' curses routine refuses to create a window that encompasses that line. Fixing such programs can be a chore, but often one can just substitute 'COLS' for 80, 'LINES-1' for 23, and so on. Rogomatic demands 24 lines so I hacked it to overlay all lines >= LINES onto line LINES-1, which is messy but tolerable. For programs beyond hope, type <*wm-esc*>z and run the program on the real terminal.

A second problem is with programs that assume a specific terminal type, such as "vt100". Wm currently understands only the pseudo-terminal type "wmvirt", so such programs cannot be run under wm. It would be nice if wm could emulate an arbitrary terminal type. Then one could run wm on an ADM terminal with one window emulating a vt100 and another emulating a Tektronix 4014. Of course, wm would need to support viewports into windows. Viewports are trivial compared to emulating 4014 graphics, however.

A third problem is that some programs use getlogin to determine the user's login name, but under wm getlogin fails since no one is logged in on the window's control terminal (pseudo-tty). The easiest fix is to replace calls to getlogin with calls to safegetlogin:

```
/*
 * Returns a user name such that uid(user name) == getuid().
 * If feasible, the session login name is used,
 * but if the real uid has been changed (e.g. via 'su')
 * or if certain file descriptors have been munged
 * then a user name corresponding to the real uid is returned instead.
 * Returns NULL if everything fails.
 * Beware!  Clobbers static data from earlier calls to getpw*.
 */

#include <pwd.h>

char *
safegetlogin()
{
        register char *p;
        register int uid;
        register struct passwd *pwd;
        static char namebuf[50];
        extern char *getlogin();

        uid = getuid();
        p = getlogin();
        /* cannot trust getlogin, so here is a security check */
        if (!p || !(pwd = getpwnam(p)) || uid != pwd->pw_uid)
                p = 0;
        /* if getlogin failed, try the real uid */
        if (!p && (pwd = getpwuid(uid)))
                p = pwd->pw_name;
        if (p) {
                strncpy(namebuf, p, sizeof(namebuf)-1);
                p = namebuf;
        }
        return(p);
}
```

This fixes the 'galaxy' and 'robots' programs, for example. We keep this routine in /usr/local/liblocal.a and link programs that call it with '-llocal'. Alas, one case that this misses is the command ''who am i''. The trade secret status of ''who'' precludes a clearer explanation.

Support for Fast Scrolling of any Rectangular Window

A (very) few terminals support general ''scrolling rectangles.'' These are wonderful for wm because then non-full-width windows scroll quickly. Terminfo already has the ''set_window'' string capability, described in 'tparm' format, to set up a scrolling rectangle. There is no such capability in termcap so a new ''sw'' capability was invented, using the same format as terminfo. For example, for the HDS Concept 108 we have:

```
:sw=\Ev%p1%' '%+%c%p3%' '%+%c%p2%p1%-%'!'%+%c%p4%p3%-%'!'%+%c:\
```

For the HDS 200 we have (sorry, this is untested):

```
:sw=\E[%p1%{1}%+%d;%p2%{1}%+%d;%p3%{1}%+%d;%p4%{1}%+%dr:\
```

You also need ''sr'' (scroll_reverse) for wm to use set_window. Wm assumes that the set_window command moves the cursor to the rectangle's home position (upper left), that cursor motion commands work in the rectangle, and that the motions are relative to the origin of the rectangle. If your terminal works differently, and it probably will, you might have to hack wm. Look for "SET_WINDOW" which, if defined,

causes code to be generated to support scroll rectangles.

LIBCURSES

Wm depends heavily on the underlying screen management software (libcurses) for correctness and efficiency. A number of changes were found to be needed in libcurses to provide correctness. The major efficiency problem has been that of providing fast scrolling. Libcurses itself does not support insert/delete line or scrolling regions to provide fast scrolling. We decided not to put such support into libcurses, because that was considered too far-reaching, so insert/delete line and scrolling regions are handled within special code in wm itself, and the libcurses windows are accordingly fixed up. Wm could support scrolling regions more efficiently (by not switching back and forth between full screen and window-sized regions on every newline). It could also handle certain non-full-width windows better. For example to scroll a window that covers all but the rightmost column it could scroll all lines involved and then redraw the rightmost column. Perhaps the termlib or other packages do that sort of thing. Also, for non-full-width windows wm could perhaps scroll two or more lines at a time, or possibly even wrap around to the top line rather than scroll.

WM and TERMINFO

Wm can be compiled with Pavel Curtis' public domain ''terminfo'' library in addition to the usual one (''Curses Classic''). During compilation the TERMINFO preprocessor variable is defined if compilation with terminfo curses is detected, so that appropriate code is generated. The terminfo conventions are nicer than the classic version so wm uses the terminfo functions and variable names, and Curses Classic is supported by redefinitions and emulation routines.

However, wm really should only be compiled with Curses Classic. The terminfo version is slower, bigger, buggier, and does not support arrow keys. Terminfo also has more fundamental problems. It does not provide support for the TERMCAP environment variable, and there is no easy way to construct a terminfo binary file analogous to the 'wmvirt' TERMCAP string. As a result wm cannot support any terminfo-compiled applications programs. When (if) this becomes a problem there will be additional incentive for wm to be able to emulate an arbitrary terminal (also see ''problems with running programs under wm'', above). Then, in conjunction with the 4.3 kernel support for window sizes, wm can dispense with the TERMCAP variable and emulate whatever terminal is desired, thus supporting both termcap and terminfo. (<*wm-esc*>t will still be needed for tip and cu.)

WM response sluggishness

An unfortunate current side-effect of wm is that certain normally responsive terminal operations are now sluggish. For example, if you cat a file and press <Interrupt> several lines will be printed before the interrupt takes effect. Other similar special characters are also handled slowly. The reason is that wm runs in raw mode, so such characters are not ''instantly'' handled in the kernel. Instead they are passed to the currently selected pseudo-tty, which then interrupts or stops or whatever. But any characters currently queued by wm for the user's terminal are still printed! (The script program has the same behavior, but it is tolerable for the normal uses of that command.) There does not seem to be a good way around this problem. Wm's current approach is to

a) Only read a few (currently 64) characters from each pseudo-tty at a time, to avoid large queues from wm to the real tty.

b) Have wm check the size of the output queue using the undocumented TIOCOUTQ ioctl, and delay reading from the pseudo-ttys if the output queue is large.

c) Have wm reduce the read and queue sizes for a while after the user types Control-S.

The first two hacks keep the response to <Interrupt> at a tolerable level. They also keep the tty high water mark from being reached, which would put wm to sleep, which would make response very sluggish indeed. The last hack gives somewhat better response to Control-S.

This approach to wm sluggishness is quite new, so more tuning and better approaches to the problem are possibilities. Also, the parameters have been tuned for a Gould 9050, which is quite a fast machine, and the resulting extra cpu involved might be intolerable on a VAX. An easy 'fix' is to comment out the

TIOCOUTQ code in wm.c (you can probably just #undef TIOCOUTQ at the top). You might want to remove the Control-S hack as well.

The '.wmrc' File

The first line of .wmrc is the prefix character. The remaining lines describe the configured windows from bottom to top. (The last line describes the window in which the user starts.) Each line consists of the window name, the number of rows and columns, and the starting row and column (zero indexed). If the number of rows (columns) is given as zero then that dimension ''flexes'' to the height (width) of the screen, which supposedly is useful when switching among terminals of different sizes.

Several users have asked for the ability to specify an alternate ''shell'' in a window. That could be done by extending the .wmrc lines to include a command to be executed in lieu of the shell. Other users have requested the ability to set the shell prompts in a window dependent way, among other things. I do not know how that might be done.

The "SNEAKYTERMCAP" Method

Ordinarily, when a window changes size, wm blasts the window with a shell command that sets TERM and TERMCAP to indicate the new window's size. (This is the same shell command generated by <*wm-esc*>t.) This not only produces clutter, it can also confuse non-shell programs running in that window. If your version of wm was compiled with SNEAKYTERMCAP defined, however, a different method is used. The TERMCAP variable is set to a filename such as /tmp/WM.33445.1 and the file contains the termcap capability string. Then, when a window changes size the /tmp file is simply rewritten. But there is a security problem with this method (or with any command that uses /tmp) and /tmp gets cluttered up with lots of ''WM'' files. It would be better if the files were kept in a subdirectory of each user's home directory, and if wm itself cleaned up dead temporary files (e.g. due to a system crash).

Browsing About in WM Windows

Some terminals have extra memory so one can look back at text that scrolls off the screen. It would be nice if wm provided that too. One hundred lines of scrolled typescript should be adequate. A plausible approach would be to type <*wm-esc*>v (for 'view') to put the window in browse mode. Then input to that window is interpreted as requests to move back and forth in the typescript, like the 'vi' scrolling commands. Any output to the real window is held pending exit from browse mode, which might be by typing '':q'', at which time the window is reset to its state on entry to browse mode. Of course, while a window is in browse mode one can still switch between windows, move, create, and kill windows, and so on.

All this can be implemented cleanly with about 100 lines of code scattered here and there (it has been written), but what about text searching and the ability to write parts of the typescript into UNIX files? What about all the other nifty 'vi' commands? Why not just run 'vi' (or your favorite viewing program) in the browse window!? Well, here is why not. Vi has to position itself at the end of the typescript in order to provide a ''seamless interface'' to wm. Okay, that's easy. We need an ''ignore first clear'' kludge so that when vi initially redraws the screen the window is not really redrawn. Uh, well, wm already has kludges. We need to suppress the vi status line (''seamless interface'' remember), say by displaying it instead in wm's status area. Er, uhm. Yuck! It could be done, but it sure would be ugly. Alas, wm does not yet support browse mode.