The journal of Apple technology.

Volume Number: 19 (2003)
Issue Number: 5
Column Tag: Section 7

X11 Tweaks

Making X11 Play Nicely on the Mac

by Rich Morin

Tweaking X11 isn't for everyone. You'll find yourself editing configuration files, rummaging through voluminous FAQs and man pages, and whining piteously to folks on assorted mailing lists. Even if you're a dedicated hacker, you may begin to wonder whether this is the way you want to spend your time.

More to the point, you may not need to tweak X11. If, like many users, you only run commands by means of the X11 "Applications" menu, adding a few commands to the list may be all you ever need to do. The syntax may be arcane, but the commands are short and the effects of a mistake are limited to the command involved.

Even if you want to run apps from the command line, there are ways to avoid hacking. If you're only running X11 apps locally (i.e., on your desktop machine), you can start them up by means of Apple's open-x11 command. This is an (undocumented) X11 analog to open(1), which starts up OSX apps. If you have X11 running, you can start up a local copy of xclock(1) by typing the following at any shell prompt:

```
open-x11 xclock
```

In short, most X11 tweaking is entirely optional. I would suggest, however, that you put /usr/bin/X11R6 into your PATH variable. This will allow you read the X11 man pages and run X11 apps without the need for open-x11.

If you're a csh(1) user, add the following to your ~/.login file:

```
setenv PATH "${PATH}:/usr/X11R6/bin"
```

If you're a bash(1) user, add the following to your $HOME/.bash_login file:

```
PATH="${PATH}:/usr/X11R6/bin"
```

By some cute trickery, described in manpath(1), the same information is to set up manual page lookups. Basically, if PATH contains a bin directory, the corresponding man directory is searched when man(1) is run.

So much for X11 tweaking, folks. That's it, all done, move along, nothing more to see here...

Still Here?

If you have more stringent requirements (or aren't entranced with Apple's "stock" X11 configuration), you may be interested in hearing about some useful "tweaks" I've run across. Even if you don't share my situation or tastes, it's useful to know how to customize X11.

Until someone provides a GUI-based configuration tool, modifying X11's behavior will require users to edit text files,

specifically, start-up scripts. The "system-wide" X11 start-up script is named /etc/X11/xinit/xinitrc; personal ones go in ~/.xinitrc. Let's walk through the system version, so you'll know which lines do what.

If xinitrc were a normal, executable script, the first line would tell the system which interpreter (i.e., /bin/sh) to use on it. In fact, things are a bit weirder than that. /bin/sh is nominally the path to sh(1), the Bourne shell; like many other vendors, however, Apple has chosen to substitute bash(1), the GNU Project's Bourne-Again SHell, running in "POSIX" mode. Normally, this substitution is invisible, but it's nice to know what's what.

In this particular case, however, the line is simply treated as a comment. Apple's X11 implementation (like many others) always uses /bin/sh to interpret its start-up files, so changing the requested interpreter wouldn't make any difference at all!

The first two characters (#!) of the script's initial line, incidentally, are commonly pronounced "pound bang" or perhaps "shebang". The second line is a comment which contains version information from rcs(1), the "revision control system" that was used in maintaining this file.

```
#!/bin/sh
# $Id: xinitrc,v 1.2 2003/02/27 19:03:30 jharper Exp $
```

The next four lines define some shell variables, used to specify file locations. $HOME is an "environment variable" which is set at login time to the user's home directory. /etc is a traditional location for Unix control files; /etc/X11/xinit, predictably, is an X11-specific subdirectory for initialization files.

```
userresources=$HOME/.Xresources
usermodmap=$HOME/.Xmodmap
sysresources=/etc/X11/xinit/.Xresources
sysmodmap=/etc/X11/xinit/.Xmodmap
```

Next, the script checks for the existence of each file, executing a specific command if the file is present. xmodmap(1) is a utility for modifying keymaps and pointer button mappings; xrdb(1) is a resource database utility.

```
# merge in defaults and keymaps
if [ -f $sysresources ]; then
    xrdb -merge $sysresources
fi
if [ -f $sysmodmap ]; then
    xmodmap $sysmodmap
fi
if [ -f $userresources ]; then
    xrdb -merge $userresources
fi
if [ -f $usermodmap ]; then
    xmodmap $usermodmap
fi
```

With all of the settings loaded, the server is in a position to start up some programs. First, it starts up an instance of xterm(1), a terminal emulator. The ampersand at the end of the line forces the program to run in the background (that is, detached from the shell process that is interpreting this script).

Finally, the script starts quartz-wm(1), the Aqua Window Manager for the X Window System on OS X. The exec command tells the current shell session to replace itself by the window manager (as opposed to waiting around for the window manager to finish). Consequently, any commands placed after the exec line will be ignored.

```
# start some nice programs
xterm &
# start the window manager
exec quartz-wm
```

Customization

As discussed in an earlier column ("X11: A C of Window Systems"), X11's goal is to provide "mechanism, not policy". This makes it highly customizable, but it also means that there are a great number of possible configurations, depending on each user's preferences and the methods chosen to achieve them. In short, don't expect Macintosh-style simplicity.

Although the system-wide control files are important to understand, I'd recommend strongly against editing them. Apple is quite likely to change one or more of these files in an update, so your changes might quietly disappear. Also, if there are any other users on your system, it might be considered bad form to give them a "non-standard" configuration.

Instead, create personal control files (~/.Xmodmap, ~/.Xresources, and/or ~/.xinitrc). If you only want to add or override some default settings, the first two of these may be sufficient. The start-up script will use any settings you put into your personal files, after the system-wide settings have been loaded.

To change X11's start-up behavior, copy /etc/X11/xinit/xinitrc to ~/.xinitrc (be sure to include the initial period!), then edit the copy. When the X11 server detects ~/.xinitrc, it will use the personal script instead of the system-wide script, so be sure to include any lines (from the original script) that you want to have run!

Rich's X11 Tweaks

Here is a personal set of tweaks, based on my own situation and preferences. They won't be the best choice for every situation, but they work well for me and show, in any case, how this sort of thing can be done.

My goal is to have a consistently Mac-like environment, allowing me to edit files and run commands on both my desktop Mac (cerberus) and a remote FreeBSD box (cfcl). I should be able to run X11 apps on either the desktop or remote machine, by means of the command line or the X11 server's "Applications" menu.

Most users run X11 apps by means of the mouse, so let's look at setting up some entries in the Applications menu. Because X11 is a network-friendly window system, we can put in either local or remotely-executed commands:

```
xclock -title cerberus
ssh -X cfcl xclock -title cfcl
xterm -bg 'pink' -geometry =80x60
```

The first two commands, above, start up instances of xclock(1) on cerberus and cfcl, respectively. The "-title" option, described in X(1), is used to indicate which machine the app is running on. The last command sets up an 80 column by 60 line xterm(1), using a garish pink background color (to distinguish it from Terminal windows).

The $PATH modifications described at the beginning of the article let me look up X11 commands and run them from the command line. Speaking of command lines, however, let's set up Terminal (yay!) as a substitute for xterm (feh!).

The default xinitrc file tells the X11 server to start up an xterm. Because this xterm doesn't have the desired characteristics (i.e., preset size and background color), I might as well get rid of it. So, I create a personal start-up file and then comment out the offending line:

```
# xterm &
```

I could, if desired, start up some number of xterm windows from this file, specifying their sizes, colors, initial positions, and many other attributes. I'm not going to do this, however, because I seldom want to run xterm at all. The Terminal works just fine for me and xterm's appearance and behavior aren't even faintly Mac-like, so why use it?

Unfortunately, this leaves me with a small problem. X11 applications use an environment variable (DISPLAY) to tell them where they should present their output (and get input such as keyboard and mouse actions). Because the xterm above was being started up by X11, this variable was being set for it automagically. How can I get DISPLAY to be set for Terminal?

If I'm willing to keep a copy of X11 running all the time, there's a really simple solution: run Terminal from the X11 start-up script! As a "child" of X11, the Terminal process will inherit DISPLAY in exactly the same way that xterm (or any other app) would. Terminal goes into the background on its own, incidentally, so we don't need an ampersand:

```
open /Applications/Utilities/Terminal.app
```

Obviously, X11 depends heavily on the DISPLAY variable; if the variable gets trashed, bad things will start to happen. Worse, the cause of the problem may not be obvious. So, I wrote a simple "sanity check", which goes into the ~/.login file on the remote machine:

```
if ($?DISPLAY) xdpyinfo > /dev/null
```

xdpyinfo(1) is a utility for displaying information about an X11 server. If DISPLAY is set at login time, this program will be run. Although the normal output is discarded, I'll find out about any DISPLAY problems from the program's error messages. (In proper Unix fashion, these are output to standard error, which is not discarded.)

X11 Forwarding

ssh(1) is the modern replacement for telnet(1). It is secure, reliable, and has some very nifty features (one of which we're about to use :-). When I log in to a remote machine, I want any X11 commands I run to display their windows on my desktop machine. This means I need a secure path for the X11 protocol (and I don't want to worry much about how it's done :-).

X11 forwarding is the "silver bullet" that solves this problem. The ssh command (on the desktop machine) exchanges some information with sshd(8), the daemon process on the remote machine. When they're done, there's a secure communication path for my session!

There are two ways to ask for X11 forwarding. Adding the "-X" option to the ssh command line is the simplest, but it only affects a single session. If you want all of your ssh sessions to do forwarding, you can add the following line to the end of /etc/ssh_config, but remember that an update may overwrite the file:

```
ForwardX11 yes
```

I decided, instead, to create an alias (in ~/.cshrc) that sets up X11-forwarded ssh sessions on cfcl:

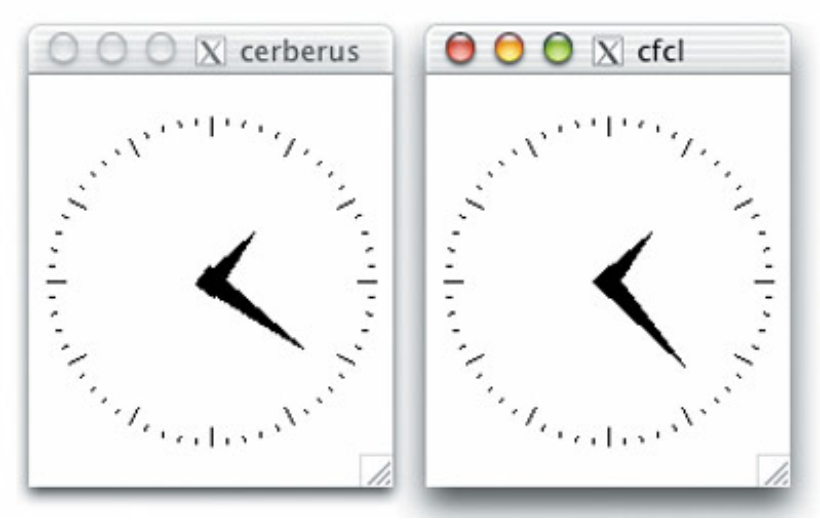
```
alias cfcl 'ssh -X cfcl'
```

In any case, I can now start up X11 (with no annoying xterm window showing up) and type commands into convenient, Mac-friendly Terminal windows. Using the "-title" option, described in X(1), I can even identify the machine an app is running on:

```
rdm@cerberus [~] 1: xclock -title cerberus &
rdm@cerberus [~] 2: cfcl
...
```

```
rdm@cfcl [~] 1: xclock -title cfcl &
```

The result, as shown in the figure, is a pair of clocks. One is running on cerberus (my Mac OS X desktop machine); the other is running on cfcl (the FreeBSD box downstairs). Both show up on my desktop, however, proving that I got everything right (except keeping the system clocks synchronized :-).



Another Approach

One objection to this approach is that it requires the X11 server to run all the time. I don't use X11 all that much, so this seems a bit heavy-handed. On the other hand, it is both simpler and safer than the method I'm about to describe. In short, proceed further at your own risk...

There are many ways to set environment variables. If you add the following line to your ~/.login file, it will get run for each new Terminal window:

```
if (! $?DISPLAY) setenv DISPLAY ':0'
```

This code is a bit subtle, so an explanation may be in order. If DISPLAY is already set (as it would be if we were coming in from an xterm), we don't want to destroy the existing setting. Otherwise, however, we're free to set up the variable.

Note: Some programs (e.g., GNU Emacs) take the existence of DISPLAY as a guarantee that an X11 server is available. If Emacs tries to talk to a nonexistent server, predictably bad things will happen. Of course, you can unset the variable before you start up Emacs, but that may seem a bit kludgy:

```
% (unsetenv DISPLAY; emacs foo)
```

Don't tell any real X11 enthusiasts that you're setting DISPLAY, by the way, unless you're ready for an extended lecture on how things are supposed to be done. For that matter, this is a klunky and somewhat fragile hack, but it works for me (:-).

Resources

I'd like to thank the patient and helpful folks on X11-users (the X11 for Mac OS X discussion list) for helping me find the information for this article. If you're planning to use X11 on OSX, be sure to join this list (http://lists.apple.com/mailman/listinfo/x11-users).

I'd also recommend that you familiarize yourself with Apple's "X11 Public Beta FAQ", the "X11: Frequently Asked Questions" thread on The MacOSXhints Forums, and the "X11 for OS X Unofficial FAQ":

http://developer.apple.com/qa/qa2001/qa1232.html

http://forums.macosxhints.com/showthread.php?s=&threadid=8704

http://www.misplaced.net/fom/X11

Even if you're not planning to do X11 development, you should grab a copy of Apple's "X11 for Mac OS X Public Beta SDK". Most X11 packages are distributed in source form; if you want to build them, you'll need the SDK!

Speaking of X11 (and other) packages for OSX, Fink is a popular and frequently recommended source for pre-compiled binaries, etc:

http://fink.sourceforge.net

Rich Morin has been using computers since 1970, Unix since 1983, and Mac-based Unix since 1986 (when he helped Apple create A/UX 1.0). When he isn't writing this column, Rich runs Prime Time Freeware (www.ptf.com), a publisher of books and CD-ROMs for the Free and Open Source software community. Feel free to write to Rich at rdm@ptf.com.