

Introduction to XFree86 4.x

Presented by developerWorks, your source for great tutorials

ibm.com/developerWorks

Table of Contents

If you're viewing this document online, you can click any of the topics below to link directly to that section.

1. About this tutorial	2
2. XFree86 4.x overview	3
3. XFree86 configuration	6
4. XFree86: the server	12
5. X client configuration	15
6. Remote access and authentication	18
7. Resources and feedback	22

Section 1. About this tutorial

Should I take this tutorial?

Perhaps you installed XFree86 4.0 (or later) and didn't know what to do next. Or perhaps you tried running X and saw nothing but a screen full of black-and-white stipple. Or quite possibly, you have X running OK, but you're sure your video card is capable of more colors or higher resolutions. In any of these cases, you've come to the right place. First, we'll get the XFree86 X server up and running for your particular video card and monitor. Then we'll cover all your personal configuration options, those little things that make your desktop feel like home.

What does this tutorial cover?

In this tutorial, Chris Houser shows you how to get XFree86 4.x, the standard free X server for Linux, up and running on your system. Chris steps you through the process of getting X configured to work properly with your hardware, and getting X running using your preferred resolution and color depth.

In addition, Chris covers X usage fundamentals, including running X applications remotely, securing X, and configuring X applications using the X resource database. He also shows you how to launch your preferred window manager, desktop environment, and applications at XFree86 startup. By the end of this tutorial, you'll have an excellent grasp of XFree86 fundamentals, and will be ready to put XFree86's many capabilities to productive use.

About the author

Chris Houser, known to his friends as "Chouser", has been a UNIX proponent since 1994 when he became the computer science network administrator at Taylor University in Indiana, where he earned his Bachelor's degree in Computer Science and Mathematics. Since then, he has gone on to work in Web application programming, user interface design, professional video software support, and now Tru64 UNIX device driver programming at [Compaq](#). He has also contributed to various free software projects, most recently to Gentoo Linux (<http://www.gentoo.org>). He lives with his wife and two cats in New Hampshire.

Chris welcomes your questions and comments on this tutorial. You can reach him by e-mail at chouser@gentoo.org.

Section 2. XFree86 4.x overview

The roles of XFree86

XFree86 is a versatile system, performing two different major roles. First we'll work with its role as a device driver; in this role, X allows your operating system and applications to communicate with your computer's hardware. To do this, XFree86 must know some details about your hardware. This information is stored in its primary configuration file, XF86Config.

Finding XF86Config

Before beginning, it's critical to find out where your XF86Config file lives. To find XF86Config, log in as root on a text console -- you shouldn't be running X right now. Typically, you'll find XF86Config in /etc/X11/; if this particular directory exists on your system, it's very likely that this is where XF86Config will be found. Other common locations include /etc/, /usr/X11R6/etc/X11/, /usr/X11R6/lib/X11/, or somewhere under /usr/lib/X11/.

If you've found where your XF86Config file *should* be, but it's not there, don't worry; I'll guide you through the steps of creating an XF86Config from scratch in just a few panels.

Skipping ahead

If you already have X working but you want to learn how to tweak various XFree86 configuration settings, skip forward to the next section, [XFree86 configuration](#) on page 6 . Now let's take a look at how to create an XF86Config file from scratch.

Creating a new XF86Config file

The easiest way to create a new XF86Config file is by using XFree86's `-configure` option. Here's how to tell XFree86 to probe your hardware and create an initial XF86Config file:

```
# XFree86 -configure
```

Your screen should go black and perhaps flash briefly -- don't let this worry you; allow a few seconds for the hardware probing to finish.

Creating a new XF86Config file, continued

If you don't see your screen go black, and instead you get a one-line error like `XF86Config: command not found`, you'll probably need to add XFree86's binary installation directory (almost always `/usr/X11R6/bin`) to your path. If you're using a Bourne-like shell (`sh`, `bash`, `ksh`, etc.), this can be done as follows:

```
# PATH="/usr/X11R6/bin:$PATH"
```

If you're running a `csh`-like shell (`csh`, `tcsh`, etc.), do this instead:

```
# setenv PATH "/usr/X11R6/bin:$PATH"
```

Once you've done one or the other, try `XF86Config -configure` again. It should work this time. Make a note to add `/usr/X11R6/bin` to your default path, so that you don't need to explicitly add it to your path every time you open up a new console.

Install your new XF86Config

Now you should have a new file named `XF86Config.new` in your home directory, `/root`. If you're replacing an existing `XF86Config` file, move it out of the way and copy this new file in its place. Of course, if there's no `XF86Config` file to replace, simply skip the first `mv` command below:

```
# cd /etc/X11
# mv XF86Config XF86Config.orig
# cp /root/XF86Config.new XF86Config
```

The first XFree86 test-run

This new `XF86Config` contains XFree86's best guess as to what kind of hardware you have and how it should be set up. To try out what we have so far, just run `XFree86` again, but this time *without* `-configure`:

```
# XFree86
```

This should again cause your screen to go black. After a few seconds, your screen should be filled with a black-and-white stipple pattern, and you should see a black X in the middle. The stipple pattern is the default XFree86 background, and the black X in the middle of the screen is the default mouse pointer. If you wiggle your mouse, it should move. If the cursor doesn't move, don't be concerned; this simply means that we need to tweak your X mouse settings, which we'll do in a bit.

To get back to your text console once the stipple pattern has appeared, hold down the Ctrl and Alt keys, and tap Backspace. This is an important key sequence to remember, because it will almost always shut down X and get you back to a text console -- just in case you can't find any other way to exit from your X session.

Test-run troubleshooting

If running `XFree86` does *not* cause a graphical stipple pattern to appear on your screen, and you get some sort of error instead, you probably have a rather unusual hardware or operating system setup that is beyond the scope of this tutorial. Now may be a good time to head over to <http://www.xfree86.org> to research whether your particular video card is supported by your release of XFree86, and if so, how to enable this support.

However, assuming all went well, you should be staring at a graphical stipple pattern. Congratulations! XFree86 is now configured with some usable defaults, but there are lots of possible configuration adjustments that can be made to make X act just the way you'd like it to act. If it's working well enough for now, though, and you're in a rush, you can skip to [Disabling remote X connections](#) on page 11; otherwise, let's load our new `XF86Config` file into a text editor.

Section 3. XFree86 configuration

A first look

Open up XF86Config with your favorite text editor, and let's see what configuration adjustments are possible:

```
# vi XF86Config
```

You should see that it's made up of several sections, each neatly set off by Section and EndSection lines. Each section contains settings for a different hardware or software component. You can see the type of component named on the Section line, such as "ServerLayout", "InputDevice", "Monitor", etc. In general, most of these sections should have been set up correctly when you ran `XFree86 -configure` earlier, and you shouldn't have to mess with them.

Configuring the mouse

If you were able to move the "X" cursor by moving your mouse when you initially tested XFree86, you can skip these next few panels and proceed to [XF86Config: depth](#) on page 7 .

However, if your mouse is not working, it means that XFree86 is using either the wrong mouse device name or the wrong mouse protocol for your particular mouse. To see the current XFree86 mouse settings, find the Sections named "InputDevice" in your XF86Config. You probably have two of these Sections, maybe more, and you want to find the one for your mouse. It shouldn't be hard to see, since the Identifier should be "Mouse0".

Configuring the mouse, continued

For mouse configuration, there are two essential configuration options: the Protocol and the Device. On modern PCs, the Protocol should almost always be "PS/2". The correct Device setting, however, may depend on your operating system, distribution, or other factors, such as whether your system uses devfs.

The proper mouse device

The most common mouse Device Option is `"/dev/mouse"`, but this only works if you have a `/dev/mouse` symlink pointing to your real mouse device name in `/dev`, and your

particular Linux system may or may not have this set up correctly. If you don't have an existing `/dev/mouse` symlink, you'll have to find the specific mouse device name yourself.

Fortunately, this process is usually quite easy. Almost all modern mice plug into your computer's PS/2 port, which in Linux shows up as the "psaux" device. This device node can be found at `/dev/psaux` or at `/dev/misc/psaux` if you happen to be using a devfs-enabled system. If one of these device nodes exist, try specifying it for the Device Option in your XF86Config; when you type `XFree86` to test X again, your mouse should start working properly.

Making the changes

The mouse InputDevice section of your XF86Config should now look something like this:

```
Section "InputDevice"
    Identifier "Mouse0"
    Driver     "mouse"
    Option     "Protocol" "PS/2"
    Option     "Device"  "/dev/psaux"
EndSection
```

If you *still* can't get your mouse working, it's possible that you don't have PS/2 mouse support enabled in your kernel. To solve this, try loading the appropriate kernel module for your mouse, or alternatively compile a fresh kernel with mouse support included.

XF86Config: depth

Now we'll take a look at the part of your XF86Config file that defined the allowed resolutions/color depth combinations for XFree86. You'll find this information in the "Screen" section, which should be at or near the end of the XF86Config file. There should be at least one (probably several) "Display" SubSections inside the "Screen" section, each with its own Depth setting.

Depth refers to the number of bits that are used to store each pixel on your screen, where a Depth of 1 is black and white, and a Depth of 24 specifies a true-color display:

Depth	Number of colors
1	2 (Black and White)
4	16 (Palette of 16)
8	256 (Palette of 256)
15	32,768 (Approximate colors)

16	65,536 (Approximate colors)
24	16,777,216 (True color mode; accurate colors)

XF86Config: depth, continued

You probably want to add a line just inside the "Screen" section to specify XFree86's default color depth, which would look something like this:

```
Section "Screen"
    DefaultDepth 24
    (rest of "Screen" section follows...)
EndSection
```

With a high Depth setting, you'll get a richer selection of colors. Generally, it's best to shoot for the highest depth that your video card will support; a depth of 15 (good) to 24 (best) is recommended since these particular depths will allow you to display photorealistic images. Anything less than a depth of 15 will cause some images to appear quite choppy, since certain colors will need to be approximated by the X server using a stippling effect known as "dithering". If you have a modern video card, you should definitely be able to go with a DefaultDepth of 24.

XF86Config: modes

You will also want to add some default resolutions to your XF86Config file. To do this, add a Modes line, like the one below, to the "Display" SubSection corresponding to the DefaultDepth setting you chose above. It's recommended to add the same Modes line to every "Display" SubSection to avoid confusion later.

```
Modes "1280x1024" "1024x768" "800x600" "640x480"
```

Each of the number pairs is a screen resolution. For example, if you're using "800x600", you will be able to see 800 pixels horizontally and 600 vertically. With smaller numbers, there is less "real-estate" on your screen, but everything that you *can* see is larger and easier to read. Note that you can't simply pick these mode names out of the blue; the ones listed above ("1024x768", etc.) are standard modes that XFree86 has been programmed to recognize. While it is possible to define your own custom resolutions, doing so requires additional steps, and you should generally stick with these predefined modes.

XF86Config: eliminating "bad" modes

An important note -- in the example on the previous panel, XFree86 will attempt to create a 1280x1024 screen, and only fall back to other resolutions if 1280x1024 doesn't happen to be supported by your video card. So if you have a DefaultDepth of 24 and a modes line like the one above for your "Depth 24" SubSection, X will try to set up a 1280x1024 resolution screen with 24-bit color by default. If, however, your video card or monitor doesn't support a resolution of 1280x1024, X will fall back to 1024x768, 800x600, etc. until a valid mode is found.

XFree86 and mode switching

However, even if a particular resolution isn't enabled by default, XFree86 still allows you to switch to it by using a special key sequence when X is running. When you're running X, you can cycle through all specified valid display resolutions by holding down the Ctrl and Alt keys and tapping the "+" and "-" keys on your numeric keypad to move forward and backward through the list.

I would recommend deleting from the list any resolutions higher than what you want to run at normally, but leaving the lower ones. The lower resolutions are sometimes helpful if you want to "zoom in" on some small icon or text on your desktop, or if you will be playing games that run best at slightly lower resolutions. Often, games will automatically switch X into a lower resolution if one is available.

XF86Config: The final "Screen" section

At this point, the "Screen" section of your XF86Config should look something like this. Don't worry if the numbers are a little different, just make sure the general order of things is similar:

```
Section "Screen"
    Identifier "Screen0"
    Device "Card0"
    Monitor "Monitor0"
    DefaultDepth 24
    SubSection "Display"
        Depth 8
        Modes "1280x1024" "1024x768" "800x600" "640x480"
    EndSubSection
    SubSection "Display"
        Depth 16
        Modes "1280x1024" "1024x768" "800x600" "640x480"
    EndSubSection
    SubSection "Display"
        Depth 24
        Modes "1280x1024" "1024x768" "800x600" "640x480"
    EndSubSection
EndSection
```

Once you've added the resolutions that you'd like to use to the XF86Config file, you can try running "XFree86" again to see the higher resolutions in action. With just a stipple pattern background, it can be tricky to notice a difference in screen resolution, but there should be a difference. You may want to come back to this part of the tutorial once you've configured X to start a few applications, which will allow you to more easily see the difference in screen resolution.

XF86Config: monitor support

As mentioned earlier, it's possible that a particular resolution selection will be ignored by XFree86, if XFree86 determines that it is not supported by your particular video card/monitor. If you have a modern "Plug and Play" monitor, XFree86 will be able to query it and determine what resolutions and refresh rates it supports, eliminating any bogus resolutions that simply won't work on your monitor. However, if you run into a situation where XFree86 will simply refuse to initialize a resolution and display depth that you *know* your monitor supports, you may need to tweak the "Monitor" section in your XF86Config file and specify your monitor's vertical and horizontal refresh settings manually. Here's how to do it.

XF86Config: monitor support, continued

First, you'll need to track down the particular refresh settings for your monitor. My particular monitor has a horizontal refresh range from 31.5 to 64.3Khz, and a vertical refresh range from 50 to 100Hz. To specify my settings manually, I've edited my "Monitor" section to look like this:

```
Section "Monitor"
    Identifier      "Monitor0"
    VendorName      "Monitor Vendor"
    ModelName       "Monitor Model"
    HorizSync       31.5 - 64.3
    VertRefresh     50 - 100
EndSection
```

After making the appropriate changes to *your* "Monitor" section, XFree86 should now start properly identifying the resolutions supported by your display.

XF86Config: other options

If you want to find out about the other settings in XF86Config, what they mean, and which of them you might want to change, see the XF86Config man page, which is quite detailed:

```
# man XF86Config
```

Remote X connections

Now that you've got XFree86's hardware configuration working, there's just one small step left before you're done with the system-wide configuration. By default, XFree86 allows client applications to talk to it a couple of different ways, including via Internet connections. This is a powerful feature, and if you intend to run some X clients remotely, you'll need this feature, and you should skip forward to the section [XFree86: the server](#) on page 12.

Remote X connections, continued

However, on many workstations, X clients are never run remotely, and having XFree86 listen to a TCP/IP port simply opens your machine up to more security risk than necessary. To disable this feature and close up this potential security hole, edit your system-wide startx script using your favorite text editor:

```
# vi /usr/X11R6/bin/startx
```

Disabling remote X connections

Startx is a shell script that we'll be talking about more in the next part of this tutorial. For now, scroll down to about line 23 at the end of the second block of text. There you should see the line:

```
serverargs=" "
```

Change this line to:

```
serverargs="-nolisten tcp"
```

XFree86 is now set up to work on your system. If you are a system administrator, this is all you need to do. However, if you are a user, you probably want to add some personal touches so that X acts the way *you* want it to act.

Section 4. XFree86: the server

Introduction to startx

The second major role that XFree86 plays is that of a server, providing central services to multiple clients. The terms "server" and "client" can seem a bit confusing when discussing X. It helps me to remember that just like a *Web* server, you generally run only one X server, and it controls access to shared resources. With a Web server, those resources may be Web pages and other files, but with X, they are your video card, monitor, keyboard, and mouse. An X client is an application or program that interacts with the server to open a window, and receives input as you click on buttons and type.

Introduction to startx, continued

Just like a Web server needs a Web client (Web browser) in order to be useful, an X server needs X clients. When you start up XFree86 with the `startx` command, the `.xinitrc` file in your home directory is read to determine what programs to launch automatically. These applications are almost always X clients, and may include terminal emulators, clocks, graphical programs, and window managers or desktop environments.

The `.xinitrc` script

The `.xinitrc` file is just a shell script, so if you're familiar with writing shell scripts, this section should make a lot of sense to you. If you've not done shell scripting, that's okay too; this will be a simple example of that worthy trade, and you'll get the hang of it quickly.

Creating a simple `.xinitrc`

Let's start with the simplest of `.xinitrc` files first. Switch to your "normal" user, instead of being the root user. Log out and back in if you have to. Then create a `.xinitrc` file in your home directory, make it "executable", and open it with your favorite text editor:

```
$ cd
$ touch .xinitrc
$ chmod a+x .xinitrc
$ vi .xinitrc
```

Then put just one line in it:

```
exec xterm
```

This tells the `startx` command to run `xterm` when it starts X. The `exec` command causes `xterm` to replace the current shell process, saving some memory.

Running startx

Now give `startx` a try:

```
$ startx
```

The screen should go black and flash like it did when you ran `XFree86` before. When you see the black-and-white stipple this time, you should also see a window with your shell prompt in it. This is the `xterm` client that you specified in your `.xinitrc`. Note that the window has no border -- no way to drag it around the screen or resize it. We'll fix that in a minute.

Exit XFree86 by logging out

`XFree86` will continue to run until your `.xinitrc` script is finished. The `xinitrc` script is not done until each program listed in it has started *and* exited. By default, `xterm` is not done until its window closes. This means that when you exit from the `xterm`, you will start a cascade effect: the `xterm` will close, so `.xinitrc` will be done, so X will shut down, and finally `startx` will finish and you will be dropped back at the console.

Give it a try. Type `exit` in the `xterm` window, and after just a moment you should end up back at your text console. This is the preferred way to exit X-windows (instead of relying on the `Ctrl-Alt-Backspace` trick).

Using a desktop environment

Now we need to add something to your `.xinitrc` so that you can move and resize your windows. There are two major ways of proceeding at this point. One is to use a desktop environment like KDE or Gnome. The other is to choose all your startup programs yourself, and maintain them in your `.xinitrc`. Starting up a desktop environment is much easier, but somewhat less flexible. I would recommend taking the second, more complicated route only if you are familiar with shell scripting.

If you want to use a desktop environment, you'll need to have it installed first. Then, to use Gnome or KDE, replace the one line in your `.xinitrc` (`xterm`) with either `exec gnome-session` or `exec startkde`, respectively. That's it! Now you can skip to the

section [X client configuration](#) on page 15.

Writing an .xinitrc shell script

If you want to script a more complicated .xinitrc, the main thing you'll need to pick is a window manager. A window manager is a special type of X client that displays "decorations" around windows, and allows you to move and resize your windows, launch programs, and do other useful things. There are many to pick from; some examples are sawfish, blackbox, enlightenment, mwm, fvwm2, and twm, but there are many others. None of these, except for twm, are included with XFree86 and will need to be installed separately if you want to use them.

Writing an .xinitrc shell script, continued

Your .xinitrc should start a window manager and probably at least one terminal emulation program, so that you have a place to type commands when X starts up. The important thing to keep in mind when you are writing your .xinitrc is that X will quit as soon as .xinitrc is done. Here is a sample .xinitrc for starting multiple X clients:

```
xterm -geometry +1+1 &  
xterm -geometry +100+100 &  
exec twm
```

As you can see, a trailing "&" is appended to the first two xterm lines; this informs the shell that the xterms should be launched in the background, allowing our .xinitrc script to proceed without waiting for each xterm to exit first. Finally, we `exec` the twm window manager. Since we use `exec` and don't use a `&`, twm will replace the current shell process, and XFree86 will not exit until twm terminates.

Section 5. X client configuration

Where we are now

At this point you should have a fully functional X setup. It is configured for your hardware, and has some basic settings customized to your preferences. You may want to take some time here to become familiar with your desktop environment or window manager. The rest of this tutorial will cover some less-necessary features of the X windowing system, such as how to customize color and font settings with the X resource database, how to run X clients on remote machines, and how to use the X authentication system. All of the instructions from here on should be run in as your "normal" non-root user in a terminal window inside X, such as `xterm`, `rxvt`, `gnome-terminal`, or `konsole`.

X client configuration

X applications can be configured in a myriad of ways. Each application may use one or a combination of the methods available. Some are unique to a specific application, and may consist of command-line options, or a configuration file in `/etc`, just like other UNIX programs. Some other configuration options may be set up through some shared external program, like the control panels for Gnome and KDE applications. Exactly *how* an X application is configured largely depends on the age of the program, whether the program was designed for a particular desktop environment, and the sensibilities of the developers who created the program. In many ways, configuration is one of the tricky aspects of X, since there is not one "official" way to do it.

The X resource database

Traditionally, X clients have stored their configuration settings in a repository called the X resource database. This database is not maintained on disk like most databases, but is stored in memory by XFree86. When an X application starts up, it scans XFree86's resource database, looking for entries that it cares about. Each entry in the X resource database is made up of a ResourceName and a corresponding Value.

Since a single database is used for storing configuration values for many different X applications, the ResourceName needs to specify not just the name of the setting, but also the name of the application to which it applies. Therefore a ResourceName specifies a hierarchy starting with the application name, and getting more specific down to the exact name of setting itself. This is a bit like the way a pathname in a filesystem starts at the top with `/` and gets more specific down to the exact file.

A demo using xrdb

For example, the background color of xterms can be set by putting an entry in the database with the ResourceName "xterm*background". We can try this out by using the `xrdb` command to add the entry to the X resources database:

```
$ echo "xterm*background: red" | xrdb
$ xterm
```

You should see a new xterm pop up with a fiery red background instead of the default white. The asterisk (*) in the middle of the name means that there may be more levels of the hierarchy between xterm and background, but that we don't care -- we want all backgrounds that are in xterms to be red. If we wanted to be more specific, we could use a period (.) instead of an asterisk, but then we would have to specify every level in between xterm and background, and we wouldn't really gain anything.

Querying the database

The `xrdb` command used in the last step can also be used to query the database. To see your current settings, you can:

```
$ xrdb -query
xterm*background: red
```

And sure enough, there is the setting you added to the database.

Adding xrdb to your .xinitrc

The normal way to use `xrdb` is to run it from your `.xinitrc`, and to use it to load your settings that you've placed in a file called `.Xresources` (or sometimes `.Xdefaults`). To set this up, add the line `xrdb $HOME/.Xresources` to the top of your `.xinitrc`, before anything else. You should do this regardless of what else is in your `.xinitrc` (that is, regardless of whether you are using a window manager directly or a desktop environment).

Create your .Xresources file

Next, create the `.Xresources` file in your home directory and put into it any of your own personal application settings. Here are a few examples (note that lines starting with a bang (!) are comments that `xrdb` ignores):

```
! use a light yellow background for everything except xterms
*background: lightyellow
xterm*background: white
! slim the widget borders on wish (TCL/TK) apps down to one pixel wide
wish*borderWidth: 1
! set the font for xterms to something a bit larger
xterm*font: -misc-fixed-**-**-20-**-**-**-*
```

Resource names and color value names

The best way to find out what X Resource settings a particular application uses is to browse the application's documentation, particularly its man page. For example, you can see the resource names that xterm recognizes by running `man xterm` and searching for the word "RESOURCE".

Any resource that specifies a color will accept any named color that XFree86 recognizes. These color names are listed in the file `/usr/X11R6/lib/X11/rgb.txt` along with a red-green-blue triplet of numbers for each. In addition to using named colors, you can also specify an HTML-style RGB hex triplet using the following syntax:

```
! specify a pure blue xterm background (#rrggbb)
xterm*background: #0000ff
```

Font names

X uses rather scary-looking strings as font names, like the one after `xterm*font` above. Don't let them worry you, though. It's easy to find the font name you want by using a font selection program. I would recommend using `gfontsel` if you have it installed, but the standard `xfontsel` also works just fine. Both programs show you an example of what the font looks like, as well as the full font name to be used in your `.Xresources` file.

Testing your `.Xresources` file

Any time you want to test out your `.Xresources` file, you can either stop and restart your X server, or just run `xrdb` again:

```
$ xrdb $HOME/.Xresources
```

Either way, changes you make to `.Xresources` won't affect any currently running programs. You'll need make sure `xrdb` loads the `.Xresources` file, and then quit any running program and start it up again to see how your new settings will affect it.

Section 6. Remote access and authentication

Remote access introduction

Unlike many graphical user interface systems, the X window system was designed with remote access in mind. It is simple to run an X client on one machine and have its windows display and be controlled by an X server running on a different machine.

To work through this section of the tutorial, you will need access to at least two different UNIX machines connected to the same network, with XFree86 installed on both. We will refer to the one you sit in front of as the *Server Machine*; the other will be called the *Client Machine*. Although both need to have X installed, only the Server Machine needs to have been configured as described so far in this tutorial. Also remember that you must *not* have "-nolisten tcp" in your startx script on the Server Machine. The Client Machine must have X libraries installed, but the X server need not be configured or running. In fact, the version of X installed on the Client Machine doesn't even need to be XFree86 -- any modern X implementation will do.

Get server hostname and display number

To get started, open an xterm (or other terminal emulation program) on your Server Machine's desktop, execute the following commands, and note the output. We'll be using this output in the next few steps.

```
$ hostname
servermachine.foo.bar.net
$ echo $DISPLAY
:0
```

You should already be familiar with the hostname command, but the DISPLAY variable may be new to you. When XFree86 starts up, it takes a display number, usually 0. We'll see how to use this display number in a minute.

Generate a new .XAuthority file

Next, you may have to generate a .Xauthority file, if you don't already have one. To find out, use the xauth command:

```
$ xauth list
xauth:  creating new authority file /home/c/.Xauthority
```

If you do not see this error message, and instead xauth lists one or more authentication lines, then you should skip this next step. An authentication line looks

something like this:

```
servermachine/unix:0 MIT-MAGIC-COOKIE-1 39849ced20eb2b62df87c714a251b8fc
```

If you *did* see the creating new authority file message, then you need to generate a new authentication key:

```
# xauth generate .
```

Shared home directory

Now, the easiest way to get a remote program's window to show up on your Server Machine's desktop is if your user's home directory is the same on both machines. This is often the case on local UNIX networks when users' home directories are mounted from some central file server via NFS or some other network file system. If this is the case with your Server and Client Machines, all you need to do is tell the Client Machine where to find your X server, covered in the "Set your DISPLAY" panels ahead.

Extracting the authentication entry

However, if the two machines do not share your home directory, you'll have to do a bit of work to give your Client the authority to talk to your Server without opening up access to the whole world. To do this, you'll need to copy the authentication entry you created earlier (or one that already existed) to your remote Client Machine. First, you must extract the entry from the Server Machine:

```
$ xauth nextract myauth.xa $DISPLAY
```

Now the file `myauth.xa` contains your X authentication data; you should be careful not to leave this file lying around where other people can see it, or to transmit it unprotected across public networks. You do need to copy it to your home directory on the Client Machine, so I would recommend using `scp` (part of the `ssh` package).

```
$ scp myauth.xa clientmachine:myauth.xa
$ rm myauth.xa
```

Merging the authentication entry

Next, log into your Client Machine using `rlogin`, `ssh`, `telnet`, or whatever you normally use to access the other machine. Then, merge your copied authentication entry into the

.Xauthority on the Client Machine:

```
$ xauth nmerge myauth.xa
$ rm myauth.xa
```

Set your DISPLAY

Your Client Machine still needs to know where to find the XFree86 server you are running. Tell it by setting the DISPLAY variable to point to your Server Machine. Make sure you are logged into your Client Machine, and use the hostname and display number you printed earlier. If you're using a Bourne-like shell, do this:

```
$ DISPLAY=servermachine.foo.bar.net:0
$ export DISPLAY
```

Otherwise, if you're using a csh-like shell, do this instead:

```
$ setenv DISPLAY servermachine.foo.bar.net:0
```

As you can see, we used your Server Machine's name, with the display number tacked on the end.

Run xeyes remotely

Now you are ready to run a program on the Client and control it from the Server Machine:

```
$ xeyes
```

You should now see a large pair of eyes (with an unhealthy interest in your mouse pointer) pop up on your screen. This program is running on a remote UNIX box (your Client Machine), and yet is interacting with your mouse and display on your local workstation (your Server Machine). To make xeyes stop, just quit the xeyes program that's running on the Client Machine by typing Ctrl-C in the terminal emulator window. The eyes will then immediately disappear from the display on your Server Machine.

Conclusion

You have now seen most of the many facets of XFree86 configuration. You've worked with several different configuration files (XF86Config, startx, xinitrc, .Xresources,

.Xauthority). You've changed remote access and authorization settings, and gotten XFree86 driver settings set up correctly. You should now have the tools you need to take control of your X windowing environment and master your UNIX desktop experience.

Section 7. Resources and feedback

Web sites

There is a lot of information on the Web about X servers in general and XFree86 in particular. Here are a few good places to start looking for more details:

- * The XFree86 home page (<http://www.xfree86.org/>) has the latest versions for download, installation notes, hardware support lists, and more.
- * The X site from the X Consortium (<http://www.x.org/>), an organization of The Open Group, distributes official releases of X (but not XFree86). Their site includes information on the X protocol, server design, etc., and is primarily of interest to experienced X developers.
- * The Direct Rendering Infrastructure (DRI) (<http://dri.sf.net/>) is a framework that provides for safe, fast OpenGL implementations, opening the door for high-speed 3D games and graphics on XFree86 for many graphics cards.
- * nVidia graphics cards are not supported by DRI; fortunately, NVIDIA provides their own high-quality accelerated XFree86-compatible drivers for Linux (<http://www.nvidia.com/view.asp?PAGE=linux>), which include a modern accelerated OpenGL implementation. This page provides access to their drivers for XFree86 on Linux.
- * Download the GNOME desktop and applications at <http://gnome.org/>.
- * The popular K Desktop Environment (KDE) (<http://www.kde.org/>) includes KOffice and the Konqueror Web browser.

Your feedback

We look forward to getting your feedback on this tutorial. Additionally, you are welcome to contact the author, Chris Houser, directly at chouser@gentoo.org.

Colophon

This tutorial was written entirely in XML, using the developerWorks Toot-O-Matic tutorial generator. The open source Toot-O-Matic tool is an XSLT stylesheet and several XSLT extension functions that convert an XML file into a number of HTML pages, a zip file, JPEG heading graphics, and two PDF files. Our ability to generate multiple text and binary formats from a single source file illustrates the power and flexibility of XML. (It also saves our production team a great deal of time and effort.)

You can get the source code for the Toot-O-Matic at www6.software.ibm.com/dl/devworks/dw-tootomatic-p. The tutorial [Building tutorials with the Toot-O-Matic](#) demonstrates how to use the Toot-O-Matic to create your own tutorials. developerWorks also hosts a forum devoted to the Toot-O-Matic; it's available at

www-105.ibm.com/developerworks/xml_df.nsf/AllViewTemplate?OpenForm&RestrictToCategory=11.
We'd love to know what you think about the tool.