The X Window User HOWTO

Christopher Yeleighton

<x11user@tldp.org>

Hal Burgiss

<hal@foobox.net>

v4.0 Jan. 12th, 2010

Revision History

Revision v4.0	2010-01-12	Revised by: cy
Converted to XML docbook and to the new domain.		
Revision v3.1	2002-10-10	Revised by: hb
Some minor additions and updates.		
Revision v3.0	2002-03-06	Revised by: hb
Rewrite of the original document. Convert to DocBook. Many, many changes.		

This document provides basic information on understanding and configuring the X Window System for Linux users. This is meant to be an introductory level document. A basic knowledge of software configuration is assumed, as is the presence of an installed and working X Window System.

Table of Contents

Table of Contents

1. Introduction

The *X Window System* is an advanced, graphical computing and network environment that was designed from the ground up as a multi-user system. X was first released in 1984. If you are not familiar with the basic concepts surrounding X and it's related components, you should first read the *X Window System Architecture Overview HOWTO*, http://linuxdoc.org/HOWTO/XWindow-Overview-HOWTO/index.html, to get an idea of how the various pieces fit together. There is also an attempt to define to various X related terminology in the Appendix, if concepts such as "displays" and "X clients" in this context are confusing to you.

This document will address basic X Window configuration and usage on Linux. We will also look at how X is commonly started in Linux, and how the start up can be configured, and related issues. We will *not* examine Window Manager (e.g. fvwm), or Desktop Environment (KDE and GNOME) configuration. There are just too many variables there, and the pace of change moves too quickly. Of course, to a large extent the user interacts more directly with these components than the X server itself, so additional reading would be worthwhile. Check your locally installed documentation, and the respective home pages for more information.

Some other important points to remember here:

- X is a client-server, multi-user system in every respect, and not just a GUI.
- X is not integrated into the operating system, and rides on top of it, like other servers.
- X is an open standard, and runs on many platforms.
- What you actually see on the screen is the result of various components, all working together: operating system, X, Window Manager, and optionally, a desktop environment like GNOME or KDE. These are all "plug and play" components, meaning you can interchange an individual component without touching the other components.
- Each of the various components has its own configuration. This makes for a very flexible, and potentially very robust, system. It also adds complexity.

The discussion here will be limited to X as implemented by The XFree86 Project, Inc. on Linux. There are other implementations, including commercial ones. XFree86 v4.x has been out for some time now, so we will be assuming that version. Much of the discussion applies to the previous 3.x version as well, but there are some occasional differences.

It is also worth noting that there are conceivably many ways to start X, and to set up a Linux system. We will focus on the common methods found in Linux distributions. Also, vendors may vary on where they put configuration files, and how they name them. Keep this in mind if you see such discrepancies in this document. If this is a problem, your vendor surely has their own documentation. And as always, hopefully the man pages will conform to your installation.

Also, we will look at various configuration files in the following sections. These are all plain text files, and can be edited with your favorite editor. Always make a backup copy before editing important files, in case Murphy pays a visit (e.g. "cp /etc/X11/XF86Config-4 /etc/X11/XF86Config-4.bak").

1.1. New Versions and ChangeLog

The current official version of this HOWTO may be found at the Linux Documentation Project.

v3.1: This is just some small, minor updates. Include link to http://www.plig.org/xwinman/ as a good resource for shopping Window Managers. Add link for fluxbox, a Window Manager with Tabbed windows. And add a brief section on improving network performance. Verify links all work.

v3.0: This is a major rewrite with several new sections. Some sections were removed, with the focus more now on just X itself (and not clients like Window Managers). New maintainer too :-)

v2.0: includes corrections from Guus Bosch, Brian J. Miller, and myself, as well as lots of new updates and info.

v1.4: include corrections and additions from Anthony J., and some very good security tips from Tomasz Motylewski.

1.2. To Do

A rudimentary troubleshooting section. Probably for v3.2.

1.3. Feedback

If you have questions or comments about this document, please feel free to email me, Hal Burgiss at `<hal@foobox.net>`. I welcome any suggestions, corrections, or additions. If you have information you would like to see in future revisions, or you would like to contribute to a future revision, please drop me a note.

1.4. Help!

I have assumed maintainership of this document because it was abandoned, and I had wanted to offer a suggested change. Well, to make a long story short, this led to a major re-write. You can help make this a better document by correcting inaccuracies, clarifying the unclear, and suggesting improvements. There is much about this topic I may not know, or not have explained well. Your help will improve this document and help other users. This document *needs* your help!

1.5. Acknowledgments

Thanks to the XFree86 development team for their efforts in providing a robust and flexible GUI. And to the whole GNU/Linux and Open Source community for making it all possible.

Also, the original author, Ray Brigleb.

Various users on comp.os.linux.x that have helped in one way or another, whether they know it or not.

Lastly, http://google.com/linux, who saved me much time with their incredible repository of information. Use it to answer questions not answered here!

1.6. Copyright

1.7. Standard Disclaimer

The information and examples given here are for illustrative purposes. Use at your own risk. Every attempt has been made to insure that the content of this document was accurate when written. If you find inaccuracies, please send me clarifications.

References to any particular company, product or brand name should not be construed as an endorsement.

---

2. XFree86

Virtually every Linux distribution comes with XFree86's X Window System implementation. This project, of course, provides us the X server, but also includes an extensive suite of utilities and applications to help implement a fully functional GUI environment.

In fact, the list would be just too long to list everything that comes with XFree86. In addition to the X server itself, here are a few of the noteworthy utilities:

xdm - the X Display Manager.

xfs - the X Font Server.

twm - a lightweight Window Manager.

xterm - the best known terminal emulator. Also, **xterm3d** and **nxterm**.

xwd - a screen and window image capturer.

xf86config - X server configuration utility.

xdpyinfo - X display information utility. This shows great detail about the X server.

xlsclients - lists currently connected X server clients.

xlsfonts - lists fonts available to X.

appres - lists the X "resources" that a program will use.

xfontsel - an application for viewing or selecting fonts.

xprop - a tool for displaying window "properties", such as the Class name of the client.

xset - sets user preferences for many things, including mouse, keyboard, sound (bell), etc.

xsetroot - a program for changing the "root window" appearance, e.g. setting a background color.

xvidtune - an application to adjust X server video modes and monitor related settings.

xwininfo - displays information about a selected "window".

xmodmap - a utility for manipulating keyboard and mouse button mappings.

Many, many fonts. And quite a bit of documentation as well.

There are many more. We'll just touch on a few of these utilities here. But feel free to explore the others. Most should have their own man pages.

2.1. Hardware

The X server controls both input (keyboard, mouse, etc) and output (display, monitor) devices.

Compatible hardware is a tough topic, since it is very much a moving target. We are forced here to avoid specifics, since this would surely change by the time you read this. And would be tediously lengthy anyway.

So let's settle for some generalities. *Most* PC type hardware is supported to one degree or another. Big help ;-)

Rule of thumb: if it is a device that uses a long-standing, commonplace protocol (e.g. PS/2), it should be well supported. Conversely, if it is something relatively new, with ground-breaking technology, the odds are not as good. This is just the nature of the beast with open source development versus manufacturers that cater more to the most popular platforms. Some manufacturers are more co-operative than others too.

Now, some general guidelines:

- Monitors - This is easy. Linux does not really need to be compatible with the monitor per se. That is the job of the video card. Any monitor that your graphics card can drive should do fine. Including, flat panel monitors.
- Video cards - This is much tougher. The X server is determined by the the chipset. Many, many are supported. But inevitably there are always some newer cards, or even revised cards, that are not. And some may have better support and better optimization than others. Advanced features such as multi-headed displays, 3D, TV out, DRI, etc., have some support as well, though this should be researched first, as the support may be limited. Supported cards are listed: http://xfree86.org/cardlist.html.

  Open source drivers are often developed incrementally. For instance, a particular card may work well for basic display purposes, but specialized features such as 3D may come much later in the development cycle. This is a quite different development model than with proprietary drivers from the manufacturer.
- Keyboards -- Any standard PC type keyboard should do fine, including PS/2, USB and many infra-red devices. Probably many "non-standard" ones too ;-)
- Mice and other pointer devices -- Most should be supported including PS/2, bus, serial, USB and many infra-red devices. Optical mice also. Unix has long preferred three button mice, though more buttons is supported as well. Many wheeled mice have X server support via the "IMPS/2" (IntelliMouse), or other specific protocols, though may require supplemental configuration for some individual applications. (See the Links section.)
- Laptops have their own unique set of problems since the hardware tends to be very specialized, and often different from what is commonly found on desktop style systems. X is supported by many. Check for details at http://www.linux-laptop.net/.

You can check the "hardware compatibility list" at your distribution's web site too. This should give a very good idea of what *should work* with your release.

Newer versions of XFree86 obviously will have better hardware support. If you are using an older Linux version and don't have full hardware support, see about upgrading XFree86. Check first to see if your distribution has updates for your release.

2.2. XF86Config

The primary configuration file for XFree86 is `XF86Config`, which may exist on your system as `XF86Config-4` for XFree86 v4.x, or possibly other variations (see man page). It is typically located as `/etc/X11/XF86Config`, though again, there may be variations in the path. If both a `XF86Config-4` and `XF86Config` exist, XFree86 v4.x will use the former. This is a required file.

`XF86Config` file defines hardware devices, and other critical components of the X server environment.

While this is a plain text file, and is editable, it is most often created during installation by whatever utility your vendor uses for this purpose. XFree86 also includes the **xf86config** utility for this, but many distributions have their own such utilities. These utilities can be run after installation if need be, to alter the configuration, or if new hardware is installed. Read your locally installed documentation first. If you attempt to hand edit this file, be sure to make a backup copy first since X will not start if this file is not to its liking ;-)

This file contains various "sections". Each section defines some fundamental aspect of XFree86, such as "InputDevice" (mouse, keyboard, joystick, etc), "monitor", or "screen". The `XF86Config` man page describes the sections and common values for each. Note that the values listed in the man page is not a comprehensive listing. There are many device specific "options". Check http://xfree86.org for notes and tips on your hardware.

The author's current `XF86Config-4`, as generated by Red Hat's installer for XFree86 4.1:

```
Section "ServerLayout"
        Identifier "XFree86 Configured"
        Screen      0  "Screen0" 0 0
        InputDevice    "Mouse0" "CorePointer"
        InputDevice    "Keyboard0" "CoreKeyboard"
EndSection

Section "Files"
   # The location of the RGB database.
   RgbPath     "/usr/X11R6/lib/X11/rgb"

   # Multiple FontPath entries are allowed (they are concatenated together)
   # By default, Red Hat 6.0 and later now use a font server independent of
   # the X server to render fonts.
   FontPath "unix/:7100"
EndSection

# Module loading section

Section "Module"
        Load  "dbe"             # Double-buffering
        Load  "GLcore"          # OpenGL support
        Load  "dri"             # Direct rendering infrastructure
        Load  "glx"             # OpenGL X protocol interface
        Load  "extmod"          # Misc. required extensions
        Load  "v4l"             # Video4Linux
#     Load  "fbdevhw"
     Load  "pex5"
     Load  "record"
     Load  "xie"
EndSection
```

```
Section "InputDevice"
        Identifier  "Keyboard0"
        Driver      "keyboard"
        Option      "XkbLayout" "us"

#    Option  "AutoRepeat"    "500 5"

# when using XQUEUE, comment out the above line, and uncomment the
# following line
#    Option  "Protocol"      "Xqueue"

# Specify which keyboard LEDs can be user-controlled (eg, with xset(1))
#    Option  "Xleds"         "1 2 3"

# To disable the XKEYBOARD extension, uncomment XkbDisable.
#    Option  "XkbDisable"

# To customize the XKB settings to suit your keyboard, modify the
# lines below (which are the defaults).  For example, for a non-U.S.
# keyboard, you will probably want to use:
#   Option  "XkbModel"      "pc102"
# If you have a US Microsoft Natural keyboard, you can use:
#   Option  "XkbModel"      "microsoft"
EndSection

Section "InputDevice"
        Identifier  "Mouse0"
        Driver      "mouse"
        Option      "Device" "/dev/mouse"
        Option      "Protocol" "IMPS/2"
        Option      "Emulate3Buttons" "off"
        Option      "ZAxisMapping" "4 5"
EndSection

Section "Monitor"
        Identifier "Sylvania F74"
        VendorName "Unknown"
        ModelName  "Unknown"
        HorizSync   30 - 70
        VertRefresh 55 - 120
        Option "dpms"
#    Modelines go here if necessary. Use xvidtune to get proper values.
EndSection

Section "Device"
        Identifier "ATI Rage 128"
        Driver "r128"
        BoardName "Unknown"
EndSection

Section "Device"
        Identifier "Linux Frame Buffer"
        Driver "fbdev"
        BoardName "Unknown"
EndSection

Section "Screen"
        Identifier "Screen0"
        Device "ATI Rage 128"
        Monitor "Sylvania F74"
        DefaultDepth 24
```

```
        Subsection "Display"
                Depth 24
                Modes "1400x1050" "1280x1024" "1152x864" "1024x768" "800x600"
        EndSubSection
        Subsection "Display"
                Depth 16
                Modes "1600x1200" "1400x1050" "1280x1024" "1152x864" "1024x768" "800x600"
        EndSubSection
        Subsection "Display"
                Depth 8
                Modes "1024x768" "800x600" "640x480"
        EndSubSection
EndSection

Section "DRI"
        Mode 0666
EndSection
```

Yours may look quite different. This is just one possible configuration with gratuitous comments from Red Hat (and me), and is for a fairly ordinary set up. There is nothing exotic here like multiple screens or displays.

It is beyond the scope of this document to explain this in detail. See the `XF86Config` man page. Also, consider visiting xfree86.org and look for specific options that might apply to your card or other hardware.

Just one quick note on the "Screen" section above. Notice there are three sub-sections, identified as "Display". Each sub-section has a different "Depth" specified, (a.k.a. ColorDepth). The "Modes" also vary somewhat according to the respective "Depth" setting. The active "Display" sub-section that will be used, is determined by the "DefaultDepth" setting (unless over-ridden by command line options). The default in this example is defined as "24", so the first sub-section will be used. Also, the highest "Mode" listed in this sub-section will be the default mode (resolution), which here is the first one listed. The first listed mode also determines the viewable screen area, which can be smaller than the mode (resolution) itself. In which case, you would have a virtual desktop that is larger than the viewable screen. To have the viewable screen, and resolution match, have the largest value as the first value listed for each "Mode".

Another note on the "Modes" here: what you see is the result of my choices during Red Hat's **Xconfigurator's** configuration. These are standard resolutions, but do not have to be! This is only limited by what your hardware can support. And you don't have to use standard width x height ratios either. Something like `1355x1112` is a valid setting (if your hardware supports it and it floats your boat!).

The X server will reject any "Modes" it thinks are invalid. You can cycle through valid modes to change screen resolution with `Ctrl-Alt-+` and `Ctrl-Alt--` (that's the keypad plus and minus keys).

In versions prior to v4.x, you would also see many "Modeline" statements that attempted to define the monitor's capabilities. These statements would look something like:

```
 # 1024x768 @ 100Hz, 80.21 kHz hsync
 Modeline "1024x768"  115.5  1024 1056 1248 1440  768  771  781  802 -HSync -VSync
```

Explicit "Modeline" definitions are not required as of 4.x ;-) This sometimes required hand editing to get optimal values in earlier versions of XFree86, though is generally not necessary with v4.x. The XFree86 Video Timings HOWTO has a nice, but rather technical, explanation of this.

If whatever configuration utility you are using, does not automatically recognize your video card or monitor specifications correctly, you are unlikely to get an optimal configuration. In such cases, you may have to manually supply the correct values. This should be available from your owner's manual (you kept that, right?). Or, check the manufacturer's web site.

Again, hand editing of this file is generally unnecessary. Should you decide this is indeed necessary, be careful. One small error may cause X to fail. Any changes to this file will require restarting X for the changes to take effect.

Using somebody else's `XF86Config` file, is generally a bad idea since they are unlikely to have identical hardware.

2.3. xvidtune and Monitor Tuning

You probably want to get the most out of your hardware. If X isn't configured optimally, consider re-running your vendor's X configuration utility and try to get better results. It is highly unlikely that you could hurt anything by experimenting. Most modern monitors now have safeguards that prevent a meltdown ;-)

If you over-do it though X may not be able to start. For this reason, I prefer to use the "startx" way of starting X (see below) while "experimenting". This way if X crashes, the display manager (GUI login) will not loop and cause you severe headaches. **startx** just gracefully goes back to a text console screen, where an error message may be visible.

Another way of tweaking monitor related settings is with XFree86's **xvidtune** program. This is run interactively and can be used to adjust various settings (see man page). The simple dialog box has sliders and buttons that allow user input and adjustment. The top part has horizontal monitor settings on the left, and vertical settings on the right. The buttons just below the sliders can be used to adjust each.

This is sometimes used to adjust the viewable screen area, such as to center it, or increase its size to fill the monitor's viewport. When **xvidtune** is launched, it defaults to the current settings.

The bottom left corner has buttons that can "Apply" new settings, "Test" new settings, or "Show" current settings (i.e. dump to screen), among other things. Any changes made here are not saved. If new settings are "Applied", it is just for the current session. Example output of **xvidtune** "Show":

```
Vendor: Unknown, Model: Unknown
Num hsync: 1, Num vsync: 1
hsync range 0:  30.00 -  70.00
vsync range 0:  55.00 - 120.00
"1400x1050"   122.00   1400 1488 1640 1880   1050 1052 1064 1082 +hsync +vsync
```

The last line is the "Modeline" being used to drive the current screen. See The XFree86 Video Timings HOWTO , for more on "Modelines".

You can test modifications, and apply them to the current session. For changes to be made permanent, they will have to be added manually to the "Monitor" section of `XF86Config` (or `XF86Config-4` for v.4.x) with a text editor.

xvidtune will dutifully warn of you of the hazards of playing with the monitor settings. It is unlikely you can hurt anything with modern monitors. But it is best used to make minor adjustments. Use at your own risk!

3. Running X

Starting an X session is typically done in one of two ways: the X session is started via a display manager (like **xdm**), and the user logs in at a GUI screen. Or, the user starts X manually after logging in to a text console. The latter is typically done with the **startx** command, which is a simple shell script wrapper for **xinit**. X runs with root privileges in either case, since it needs raw access to hardware devices.

Typically, which method is used, is determined by the system "runlevel". The default runlevel to launch at boot is generally set in `/etc/inittab` on Linux:

```
# Run xdm in runlevel 5
x:5:respawn:/etc/X11/xdm -nodaemon
```

That would start **xdm**, and thus X, at runlevel 5. It will "respawn", if it dies or is stopped for any reason. You can also use the "**init**" command to change runlevels without rebooting (see man page).

Let's look briefly at both approaches, and then some additional configuration to set up the user's working environment.

3.1. startx

startx will start X by first invoking **xinit**. By itself, this would put you at a blank, fuzzy looking, bare-bones desktop with no Window Manager loaded. **xinit** basically takes two sets of command line arguments: client specifications (programs to run, etc), and server specifications (X server options), separated by "--". If no client program is specified on the command line, **xinit** will look for a `.xinitrc` file in the user's home directory, to run as a shell script. If found, this then would in turn run whatever user specified commands to set up the environment, or launch programs that the file contained. If this file does not exist, **xinit** will use the following initial command:

```
xterm -geometry +1+1 -n login -display :0
```

If no `.xserverrc` is found in the user's home directory, X itself will be started with the following command:

```
X :0
```

As you see, this is not overly helpful as it just launches one xterm. The **startx** shell wrapper provides additional functionality and flexibility to **xinit**. **startx** will invoke **xinit** for us, and provide some simple configuration options as well. You can also issue commands such as the following, for instance:

```
startx -- -dpi 100 -depth 16   #force X to 100 dots per inch
                               #and colordepth of 16 (X v4 syntax)
```

Anything after the double dashes are passed as arguments directly to the X server via **xinit**. In this example, you can force X to the resolution of your preference, and still have it use the configuration files we will cover later in this document. See the Xserver man page for more command line options.

Instead of issuing the same command line every time, it is easier to use the configuration files to store this type of information for us.

If you take a look at the **startx** script (`/usr/X11R6/bin/startx` on my system), you see it uses two default configuration files to help set up the X environment: `xinitrc` and `xserverrc`. It looks first in `/etc/X11/xinit/`, for the system wide files. It then checks the user's home directory for similar files, which will take precedence if found. Note that the latter are Unix style "dot" files (e.g. `~/.xinitrc`), and are executable shell scripts.

You normally would not want to edit the system wide files, but you can freely copy these to your home directory as a starting point, or just start from scratch. As you can tell by the names, one helps set up the X server, and one sets up **xinit** by executing commands, preparing the environment and possibly starting client programs like **xterm** or a Window Manager (yes, it's a client too).

3.1.1. xserverrc

As with all XFree86 configuration files, this is a plain text file, and is usually a simple, one line statement to start the X server. It can include any valid command line options supported by your X installation. If you always start X with your own options, this should be easier than typing the options each time. One possible `~/.xserverrc`:

```
 exec X :0 -dpi 100 -nolisten tcp
```

This will start X on display :0, the first "display", at a dots-per-inch resolution of 100, and disables TCP connections. See the Xserver man page for other valid options. This is just an example.

3.1.2. xinitrc

`xinitrc` is used to set up a suitable X environment, and to launch other programs, a.k.a "clients" that we may want available as soon as X is started. You likely have a system wide `xinitrc` to start a predefined set off programs. To customize this, create your own in your home directory. Name it `.xinitrc`, make sure it is an executable script, and **chmod +x**. An example (slightly modified from the original on my system):

```
#!/bin/sh
# $XConsortium: xinitrc.cpp,v 1.4 91/08/22 11:41:34 rws Exp $

userresources=$HOME/.Xresources
usermodmap=$HOME/.Xmodmap

# merge in defaults and keymaps
if [ -f $userresources ]; then
    xrdb -merge $userresources
fi

if [ -f $usermodmap ]; then
    xmodmap $usermodmap
fi

if [ -z "$BROWSER" ] ; then
        # we need to find a browser on this system
        BROWSER=`which netscape`
        if [ -z "$BROWSER" ] || [ ! -e "$BROWSER" ] ; then
        # not found yet
```

```
                BROWSER=
        fi
fi
if [ -z "$BROWSER" ] ; then
        # we need to find a browser on this system
        BROWSER=`which lynx`
        if [ -z "$BROWSER" ] || [ ! -e "$BROWSER" ] ; then
        # not found yet
                BROWSER=
        else
                BROWSER="xterm -font 9x15 -e lynx"
        fi
fi

export BROWSER

# start some nice programs
if [ -f $HOME/.Xclients ]; then
    exec $HOME/.Xclients
else
    xclock -geometry 50x50-1+1 &
    xterm -geometry 80x50+494+51 &
    if [ -f /usr/X11R6/bin/fvwm ]; then
        exec fvwm
    else
        exec twm
    fi
fi

#eof
```

Briefly, what this script does, is set up our working environment, with **xmodmap** (keyboard) and **xrdb** (application resource settings). More on these below. Then the shell variable `$BROWSER` is set for a GUI environment (Netscape in this example) so that any applications that might expect this, have a reasonable choice available. Then the presence of the file `Xclients` is checked, both as a system wide file and in the user's home directory. In this particular example, this is where any client applications are to be started, including a Window Manager (see below). These could just have as easily been started here if we had wanted to. If an `Xclients` file can't be found, then a Window Manager is started for us. Either **fvwm**, if available, or XFree86's minimalist **twm** if not. If for some reason, neither of these can be started, the script would exit, and X would fail to start.

3.1.3. Xclients

Everything up to this point has followed pretty much a standard and predictable sequence of events. To summarize, we have invoked **startx**, which in turn invoked **xinit**, which has parsed `xinitrc` for initial settings. Most Linuxes should follow this same sequence, though the various values and settings may differ.

We now are at the last link in the chain where the user normally would specify his or her preferences, including the Window Manager and/or desktop environment to be used. The system will provide sane, though possibly uninteresting, defaults if the user has not done so. Presumably, this is why you are here ;-)

The Window Manager, or desktop environment, is typically the last application started. If you want other programs (like **xterm**) started, they should be started before the Window Manager and "backgrounded" with an "&". This can all be done in the user's `~/.xinitrc`. Or as in the above example, the actual applications are started from yet another script. Let's look at one short, hypothetical such script, `.Xclients`:

```
#!/bin/bash
# ~/.Xclients, start my programs.

xset s off s noblank
xset m 30/10 4
xset r rate 200 40

xscreensaver &
rxvt -geometry 80x50-50+150 &

echo Starting Window Manager...

if [ -x /usr/X11R6/bin/wmaker ]; then
  echo `date`: Trying /usr/X11R6/bin/wmaker... |tee -a ~/.wm-errors 2>&1
  exec /usr/X11R6/bin/wmaker >> ~/.wm-errors 2>&1
fi

echo `date`: Failed, trying fvwm... |tee -a ~/.wm-errors 2>&1

# let's try regular fvwm (AnotherLevel doesn't work with fvwm1).
if [ -n "$(type -path fvwm)" ]; then
  # if this works, we stop here
  exec fvwm >> ~/.wm-errors 2>&1
fi

echo `date`: Failed, trying twm... |tee -a ~/.wm-errors 2>&1

# wow, fvwm isn't here either ...
# use twm as a last resort.
exec twm >> ~/.wm-errors 2>&1

# Dead in the water here, X will exit as well, sigh...
echo `date`: Unable to start a Window Manager ... |tee -a ~/.wm-errors 2>&1

# eof
```

This really isn't so different than what `xinitrc` was doing at all. We added a few wrinkles, including starting a screen saver, a different terminal emulator that this user prefers (**rxvt**), with even more setting up of the environment (monitor, mouse and keyboard) using **xset** this time, and a different Window Manager than was available with the system defaults. This is in the user's home directory, so it will not be overwritten during upgrades too.

Actually, X has already started at this point, and we are just putting the finishing touches on the configuration. Notice the Window Managers are not "backgrounded" with "&" here. This is important! Something has to run in the foreground, or X will exit. We didn't start a desktop environment in this example, like KDE or GNOME, but if we did, this final application would have to be **gnome-session** or **startkde** instead. Since we are rolling our own here, if we wanted to change Window Managers, all we have to do is edit this file, and restart X. Vendor supplied configurations may be more complex than this, but the same principles apply.

As an afterword, do not think that any initial client applications *must* be started as we've done here. This is how it has been traditionally done, and some may prefer this approach. Most window managers have their own built-in ways to start initial programs, as do KDE and GNOME. See the respective documentation.

3.2. Display Managers

The other, more common, approach is the "GUI log-in", where X is running before log-in. This is done with the help of a "display manager", of which there are various implementations. XFree86 includes **xdm** (X Display Manager) for this purpose, though your distribution may use one of the others such as **gdm** (GNOME) or **kdm** (KDE).

Display managers really do much more than enable GUI style log-ins. They are also used to manage local as well as remote "displays" on a network. We shall not get into details on this here, but it is nicely covered in the *Remote X Apps Mini HOWTO* and the *XDMCP HOWTO* (see the links section). For our purposes here, they provide similar services to **getty** and **login**, which allow users to log into a system and start their default shell, but in a GUI environment.

Here is an example of a more advanced usage of what else a display manager might be used for, from Diego Zamboni:

> I have two X sessions running with different resolutions. I switch between them depending on whether my laptop is connected to an external monitor or using its own LCD display.
>
> Here's my `/usr/lib/X11/xdm/Xservers` file that initiates both displays:
>
> ```
> :1 local /usr/X11R6/bin/X :1 -layout 1024x768
> :0 local /usr/X11R6/bin/X :0 -layout 1600x1200
> ```
>
> Then I have "1024x768" and "1600x1200" defined as "server layouts" in my `/etc/X11/XF86Config-4`, as follows:
>
> ```
> Section "ServerLayout"
>         Identifier     "1600x1200"
>         Screen         "Screen0" 0 0
>         InputDevice    "Mouse0" "CorePointer"
>         InputDevice    "Keyboard0" "CoreKeyboard"
> EndSection
>
> Section "ServerLayout"
>         Identifier     "1024x768"
>         Screen         "Screen1" 0 0
>         InputDevice    "Mouse0" "CorePointer"
>         InputDevice    "Keyboard0" "CoreKeyboard"
> EndSection
>
> ## snip ...
>
> Section "Screen"
>         Identifier   "Screen0"
>         Device       "S3 Savage/MX"
>         Monitor      "Monitor0"
>         DefaultDepth 16
>
>         Subsection "Display"
>                 Depth  16
>                 Modes  "1600x1200" "1280x1024" "1024x768"
>         EndSubsection
> EndSection
> ```

```
Section "Screen"
        Identifier   "Screen1"
        Device       "S3 Savage/MX"
        Monitor      "Monitor0"
        DefaultDepth 16

        Subsection "Display"
                Depth  16
                Modes  "1024x768" "800x600"
        EndSubsection
EndSection
```

Note the use of "Identifiers" here. Diego is starting two separate "displays" here. Then he can choose which one he wants when he logs in.

Most display managers are derived from XFree86's venerable **xdm**, and add their own enhancements. Let's look at the most popular ones briefly.

3.2.1. xdm

xdm can be configured with configuration files located in `/etc/X11/xdm/`, `/usr/X11R6/lib/X11/xdm`, or similar locations depending on your system. These are system wide files. The file `xdm-config` is the main configuration file, and mostly describes where to find secondary configuration files:

```
! $XConsortium: xdm-conf.cpp /main/3 1996/01/15 15:17:26 gildea $
DisplayManager.errorLogFile:   /var/log/xdm-errors
DisplayManager.servers:                /etc/X11/xdm/Xservers
DisplayManager.accessFile:     /etc/X11/xdm/Xaccess
! All displays should use authorization, but we cannot be sure
! X terminals will be configured that way, so by default
! use authorization only for local displays :0, :1, etc.
DisplayManager._0.authorize:   true
DisplayManager._1.authorize:   true
! The following three resources set up display :0 as the console.
DisplayManager._0.setup:       /etc/X11/xdm/Xsetup_0
DisplayManager._0.startup:     /etc/X11/xdm/GiveConsole
DisplayManager._0.reset:       /etc/X11/xdm/TakeConsole
!
DisplayManager*resources:      /etc/X11/xdm/Xresources
DisplayManager*session:                /etc/X11/xdm/Xsession
!
! SECURITY: do not listen for XDMCP or Chooser requests
! Comment out this line if you want to manage X terminals with xdm
DisplayManager.requestPort:    0
```

The "!" denotes comments. The command that starts the X server is in `/etc/X11/xdm/Xservers` in this particular example as defined by "DisplayManager.servers", and is the equivalent to `xserverrc` that was used for **startx** X server start up commands, but the syntax is slightly different here. The contents of `/etc/X11/xdm/Xservers` on my system are simply:

```
:0 local /usr/X11R6/bin/X
```

This starts X on the first local display (designated by 0). Any special command line arguments that you want to add go here at the end.

Below is a sample `/etc/X11/xdm/Xsetup_0` which is used to configure the log-in screen only. Notice that we're using a shell script here, and it's calling **xv** (a graphics display program) to set the background to a nice image (instead of the boring black and white background pattern), and if that fails, **xsetroot** is then invoked to at least try to set the background to a nicer blue color. This does not configure the login widget itself -- just other things that might be wanted on the screen during login.

```
#!/bin/sh
xconsole -geometry 480x100-0-0 -daemon -notify -verbose -fn \
  '-schumacher-clean-medium-r-*-*-10-*-*-*-*-*-*-*' -exitOnFail  &

/usr/X11R6/bin/xv -quit -root /usr/share/pixmaps/Backgrounds/InDreams.jpg \
  || xsetroot -solid darkblue
```

`/etc/X11/xdm/Xresources` controls the X "resources" used during log in. In this context, "resources" are user preferences for such items as fonts and colors (described in more detail below). Below is a snippet that sets up fonts for the log-in widget:

```
#if WIDTH > 800
xlogin*greetFont: -adobe-helvetica-bold-o-normal--24-240-75-75-p-138-iso8859-1
xlogin*font: -adobe-helvetica-medium-r-normal--18-180-75-75-p-103-iso8859-1
xlogin*promptFont: -adobe-helvetica-bold-r-normal--18-180-75-75-p-103-iso8859-1
xlogin*failFont: -adobe-helvetica-bold-r-normal--18-180-75-75-p-103-iso8859-1
#else
xlogin*greetFont: -adobe-helvetica-bold-o-normal--17-120-100-100-p-92-iso8859-1
xlogin*font: -adobe-helvetica-medium-r-normal--12-120-75-75-p-69-iso8859-1
xlogin*promptFont: -adobe-helvetica-bold-r-normal--12-120-75-75-p-69-iso8859-1
xlogin*failFont: -adobe-helvetica-bold-o-normal--14-140-75-75-p-82-iso8859-1
#endif
```

As you can see this is using helvetica as the preferred font, with different point sizes and dots per inch depending on the screen size. This is customizable to suit individual needs. (See below for more on understanding X font naming conventions.) Various other aspects can similarly be configured.

`/etc/X11/xdm/Xsession` is the rough equivalent to `xinitrc` for **startx**. It will similarly set up a default environment for keyboard, etc. And can also start either KDE or GNOME, and other X client programs. This is the system wide configuration file. It should also check the user's home directory for `~/.xsession`, and possibly `~/.Xclients`, which would contain the user's preferred environment and start up programs, just as `~/.xinitrc` did with **startx**. Again, the files in a user's home directory may be created or modified by the user any time and must be executable shell scripts.

We shall not include an `~/.xsession` example here, since it would be very similar to the `~/.xinitrc` and `~/.Xclients` examples above.

We've looked only briefly at the main **xdm** configuration files. Be sure to read the man page, and look at what is installed locally, for more information. Let's look now at **gdm** and **kdm**. We'll just highlight significant differences, since they essentially provide the same functionality.

3.2.2. gdm

gdm is the default display manager for GNOME. **gdm** was written from scratch, but functions similarly to **xdm**. The main configuration file is `gdm.conf`, typically located as `/etc/X11/gdm/gdm.conf`. This is quite different looking than `xdm-config`. Comments are denoted with a "#", and the file has sections, with section headers enclosed in square brackets. The command to start X is in the "[servers]" section:

```
[servers]
0=/usr/bin/X11/X
#1=/usr/bin/X11/X
```

Notice this has potentially two displays set up, but the second one is commented out. Add any additional X startup options here, e.g. "-dpi 100". The log-in screen and log-in widget are configured in the "[greeter]" section.

Start up clients and programs are determined by the "SessionDir" statement in the "[daemon]" section. On my installation, this points to `/etc/X11/gdm/Sessions/`, which contains several short scripts. If I look at my `Default` script, it actually executes `/etc/X11/xdm/Xsession`, which in turn would execute `~/.xsession`, if present. So at this final stage, **gdm** acts very much like **xdm**.

GNOME includes the **gdmconfig** utility to control many aspects of **gdm** behavior.

3.2.3. kdm

kdm is the display manager from KDE. The main configuration file for **kdm** is `kdmrc` and is typically installed as `/etc/kde/kdm/kdmrc`. As is the case with `gdm.conf`, `kdmrc` uses "#" for comments, and has sections with section headers in similar square brackets. **kdm** configuration can also be edited with the **kcontrol** utility.

The visible desktop is configured in the "[Desktop*]" section(s), and by the "Setup" directive which should point to a file like `/usr/share/config/kdm/Xsetup` or `/etc/X11/xdm/Xsetup_0`. This will accomplish the same thing as **xdm**'s `Xsetup_0` does: namely running any programs the user might want such as **xconsole**.

The command to launch the X server is the "Xservers" directive in the "[General]". Again, this should point to a file such as `/etc/X11/xdm/Xservers`, and uses the same syntax as **xdm**:

```
:0 local /usr/X11R6/bin/X
```

Any command line options for the X server, go here.

The login widget itself is configured in the "[X-*-Greeter]" section(s). Compiled in defaults are used if the user does not specify any.

KDE includes the **kdmdesktop** utility to control some aspects of **kdm** behavior, mostly just the login background.

4. More X Configuration

Before taking a look at various configuration mechanisms for X servers and clients, it should be noted that the advent of Desktop Environments like KDE have become popular in part because they can control much of the user interaction configuration themselves with nice, "user friendly" GUI controls. And in fact, the compliant applications that are part of the respective Desktops will be best configured through the Desktop's configuration tools, or the application's own GUI configuration methods. So, for instance, **gtop**, a GNOME client application, is best configured via GNOME or **gtop's** own menus. But this is not true of all X applications.

4.1. X Resources

The X server can store various configuration values for client programs so they are readily available when needed. If the application supports this, it will use these as defaults whenever that program is invoked. These are known as "Resources", and are often used to define user preferences on a per application basis for fonts, colors, screen placement (geometry) and various other attributes. This makes it easy to customize applications.

Resources are specified as text strings (e.g. Netscape*blinkingEnabled: False) that can be read from disk in various places when X is starting, or even interactively defined on the command line. Program components are named in a hierarchical fashion, with each object in the hierarchy identified by a class as well as an instance name. At the top level of the hierarchy is the class and instance name of the application itself. Typically, the class name of the application is the same as the program name, but with the first letter capitalized (e.g. Vim or Emacs) although some programs that begin with the letter "X" also capitalize the second letter for historical reasons (e.g. XTerm). Each definition will specify a class (or instance), with corresponding resource and value. Below this in the hierarchy are the various attributes that make up the definable aspects of the application.

Traditionally, most X programs were configured this way. This is not as true today with the advent of Desktop Environments which often have their own configuration mechanisms.

As an example, say we prefer to run **xterm** with a blue background. So if we run it from the command line, we would run it as:

```
xterm -bg blue &
```

If this is our preference, it would be easier to put this preference in a file somewhere, and have the system use our preference. That way whenever we started **xterm**, it would use our preferred value, and we wouldn't need the command line options (unless as an override).

The basic X resource syntax is expressed like:

```
<program><binding><widget><binding><widget><...><resource>:<value>
```

Which, in real life, typically looks something like:

```
xterm*fontMenu*background: darkblue
```

It should be obvious what this does. The use of "*" in the definition, is called a "loose binding" and acts as a wild-card. Meaning there may be gaps in the widget hierarchy. For instance:

```
xterm*background: darkblue
```

This would also give a dark blue background for the **xterm** fontMenu, but also any other **xterm** properties that also have a "background" attribute (e.g. window background, etc), no matter where they may be in the widget hierarchy. Similarly:

```
*background: darkblue
```

This would define the background for any and all programs that support it -- not just **xterm**. Using a "." in place of a "*" would be more precise, and will not allow for wild-card gaps in the hierarchy. Also, the application must support the particular widget attribute. "Background" is a fairly safe bet, but many applications will have more specialized resources that are not so obvious. It is best to check local documentation (man pages, etc), or see if an application has an included examples. For instance, **Netscape** generally comes with an `Netscape.ad` file that has an extensive set of resource definitions that can be customized.

X resources are typically stored in more than one place (see below) and are processed by the **xrdb** command (see man page).

4.1.1. App Defaults

One way of storing preferred application resources is via files named for the application in an "app-defaults" directory. For instance, on my system, these are in `/usr/X11R6/lib/X11/app-defaults/`, though this may vary according to options your vendor has chosen. This directory contains a number of files for such well known X applications as **xterm**, **xclock**, **xcalc**, **xload**, and so on. All in all, it is a relatively small number of applications in the overall scheme of things. So not all applications use this scheme. In fact, most do not.

Each file will contain resource definitions for that application. The X server loads these by itself during start up. A brief example from `XTerm-color`:

```
! $XFree86$

#include "XTerm"

*VT100*colorMode: on
*VT100*dynamicColors: on

! Uncomment this use color for underline attribute
!*VT100*colorULMode: on
!*VT100*underLine: off

! Uncomment this to use color for the bold attribute
!*VT100*colorBDMode: on

*VT100*color0: black
*VT100*color1: red3
*VT100*color2: green3
*VT100*color3: yellow3
```

```
*VT100*color4: blue3
*VT100*color5: magenta3
*VT100*color6: cyan3
*VT100*color7: gray90
*VT100*color8: gray30
*VT100*color9: red
*VT100*color10: green
*VT100*color11: yellow
*VT100*color12: blue
*VT100*color13: magenta
*VT100*color14: cyan
*VT100*color15: white
*VT100*colorUL: yellow
*VT100*colorBD: white
```

This is mostly various color definitions. The application classname is not explicitly stated, and is assumed from the filename. So think of each line as starting: `XTerm-color*`. Also, notice at the top, the `#include "XTerm"` line, which "includes" the resource definitions for **XTerm**, a much longer file with a more diverse set of definitions. (Not included due to length, but worth looking at.) These files provide system wide defaults, and generally speaking, would not normally be edited by the user.

4.1.2. Xdefaults

Another common method of reading in resource preferences, is with an `Xdefaults` file. Or, sometimes the naming scheme may be `Xresources` instead. This may exist as a system wide file, such as `/etc/X11/Xresources`. Of course, the user is free to create a personal version in his home directory, e.g. `~/.Xdefaults`. The user's version will over-ride any system wide settings, and will remain after system upgrades. Obviously, this is the place to put your own preferences.

`Xresources` files are read into the resource database with the **xrdb** command. Example:

```
xrdb -merge ~/.Xresources
```

This can be done interactively at the command line, or placed in a script and run automatically as the X session is started. In the case of system wide files, this should be taken care of by the vendor supplied start up scripts. Generally, such scripts will also check the user's home directory as well (see the `xinitrc` example above). So probably all that need be done, is to create the file with a text editor.

Here's an example to illustrate a very few of the many things that might be done with an `.Xdefaults` file:

```
! This is a comment ;-)

#ifdef COLOR
*customization: -color
#endif

!! Let's cast a wide net, for any app supporting these
! Blink instead of beeping
*visualBell: True
*scrollTtyOutput: False
*scrollKey:       True

! See Netscape.ad for many settable resources
Netscape*noAboutSplash:        True
```

```
Netscape*documentFonts.sizeIncrement: 5
Netscape*documentFonts.xResolution*iso-8859-1: 120
Netscape*documentFonts.yResolution*iso-8859-1: 120
netscape-navigator*geometry: 960x820+240+140

emacs*Background: DarkBlue
emacs*Foreground: Wheat
emacs*pointerColor: Orchid
emacs*cursorColor: Orchid
emacs*bitmapIcon: on
emacs*font: 10x20

! GVim colors, etc
!! GTK versions of gvim will not use all these.
Vim*useSchemes:         all
Vim*sgiMode:            true
Vim*useEnhancedFSB:     true
Vim.foreground:         Black
!Vim.background:        lightyellow2
Vim*background:         white
! geometry: width x height
Vim.geometry:  88x40
Vim*font:  -misc-fixed-medium-r-normal--20-200-75-75-c-100-iso8859-15-*5
Vim*menuBackground: yellow
Vim*menuForeground: black

rxvt*backspacekey: ^?
rxvt*background: Black
rxvt*foreground: wheat
rxvt*cursorColor: Orchid
rxvt*geometry: 100x18+40+300
rxvt*title: Linux
rxvt*reverseVideo: false
!rxvt*backgroundPixmap: ~/penguinitis.xpm
rxvt*scrollBar: true
rxvt*reverseWrap: true
rxvt*font: -*-lucidatypewriter-medium-*-*-*-14-*-*-*-*-*-*-*
rxvt*fullCursor: true
rxvt*saveLines:  1500
rxvt*menu: ~/rxvt.menu

XTerm*saveLines:    1500
! Do not clear the screen after the program exits
XTerm*VT100*titeInhibit: true

! Fix up xterm's keybindings
xterm*VT100.translations:      #override \
      <Key>BackSpace:    string(0x7F) \n\
      <Key>Insert:       string(0x1b) string("[2~")\n\
      <Key>Delete:       string(0x1b) string("[3~")\n\
      <Key>Home:         string(0x1b) string("[1~")\n\
      <Key>End:             string(0x1b) string("[4~")\n\
      <Key>Page_Up:      string(0x1b) string("[5~")\n\
      <Key>Page_Down:    string(0x1b) string("[6~")\n\
      <KeyPress>Prior : scroll-back(1,page)\n\
      <KeyPress>Next : scroll-forw(1,page)

! Ghostview
Ghostview*Font: *-helvetica-bold-r-normal--12-*-*-*-*-*-*
Ghostview*BorderColor:  white
Ghostview*Text*Font:    rk14
Ghostview*Background:   #d9d9d9
```

```
!Ghostview*Foreground:  white
ghostview.form.pageview.page.background: white
ghostview.form.pageview.page.foreground: black
.ghostview.zoom.form.page.background: white
.ghostview.zoom.form.page.foreground: black

! xscreensaver !
! Time out after 12 minutes, cycle mode after each 2
xscreensaver.timeout: 12
xscreensaver.cycle: 5
! Run low priority, and fade between modes
xscreensaver.nice: 12
xscreensaver.fadeSeconds: 2

XFontsel.menu.options.showUnselectable: False
```

Hopefully, these few examples will give you some ideas to build on. X does not need to be restarted if **xrdb** is used interactively from the command line after making changes. The effects are immediate.

Resources are sometimes available also as command line options. See below. Command line options will over-ride any existing resource definitions.

4.2. xmodmap, the Keyboard and Mice

The keyboard and mouse, as well as other possible input devices, are defined in `XF86Config` (or `XF86Config-4`). There is a keyboard layout that is defined based on the preferred language:

```
Section "InputDevice"
       Identifier  "Keyboard0"
       Driver      "keyboard"
       Option      "XkbLayout" "us"
EndSection
```

This gives us our default keyboard layout. Valid layout labels are listed in `/usr/X11R6/lib/X11/xkb/symbols`. Also, the **setxkbmap** utility can be used to change this interactively.

X is highly customizable, and we can modify the keyboard and mouse pointer mappings to suit our own preferences. The utility to do this is **xmodmap** (see man page). You don't like where the capslock key is? So move it ;-)

Like **xrdb**, **xmodmap** can be run from the command line. Or, preferred settings can be stored in a file. Typically this is `~/.Xmodmap`, or similar. If your X start up files don't parse this, then edit as appropriate so that they do (probably from `~/.xinitrc` or `~/.xsession`).

You can view your current key and mouse mappings with: **xmodmap -pk -pp |less**. This will print out all active "keycode" values, with corresponding "keysym" values, and any keysym names that **xmodmap** knows about (e.g. "BackSpace"). And should also give you an idea of how **xmodmap** understands key and mouse events. There are two keysyms per keycode. The second is the shifted value. XFree86's **xev** utility can be used to dump a lot of information on key-presses and mouse events interactively. Pay attention to the "keycode" value. That is what you will need to know in order to re-map.

xmodmap is often used to make minor keyboard adjustments, like proper Backspace/Delete mapping. Or can be used make major adjustments such as for international mappings. You can only re-map keys and mouse events -- you cannot assign macros to key events (your Window Manager or Desktop might have some of this functionality).

A nice discussion of setting up international keyboards . Also, Google search will turn up many creative examples.

The man page has many brief examples of various usages. Here is what an one hypothetical `~/.Xmodmap` might look like:

```
! /home/hal/.Xmodmap, last change 10/03/01
!
! Force backspace to 22 and Delete to 111
keycode 22 = BackSpace
keycode 111 = Delete
!
! My keyboard handles right and left Alt differently. Make the
! Right act like the Left to avoid digital gymnastics.
keycode 63 = Alt_L
keycode 113 = Meta_L
!
! Hard-code the keypad to numeric values as if numlock is always on
! since I never use it for anything else.
keycode 79=7
keycode 80=8
keycode 81=9
keycode 83=4
keycode 84=5
keycode 85=6
keycode 87=1
keycode 88=2
keycode 89=3
keycode 90=0
keycode 91=period
keycode 86 = plus
! deactivate Num_Lock key since we don't need it now.
keycode 77 =
!
! My capslock is next to tab. I hit it by mistake sometimes,
! and don't use it anyway. So make capslock act like Tab.
keycode 66 = Tab
clear lock
!
! Reverse mouse buttons for left-handed people
pointer = 3 2 1
```

As with many XFree86 files, the "!" represents a comment. Another possible use, is to redefine those annoying "Windows" keys to something useful. Hopefully this gives an idea of some things one might want to do to make the keyboard more agreeable to us.

Speaking of the `Numlock` key, X will typically disable this when it starts up. No matter how you have the BIOS set up, or Linux set up before X starts. So the trick above is one way. There is also a utility available as either numlockx, or setnumlock, that can be found on the â— Net, if your distribution does not include one or the other. This can be put in a start up file to turn `Numlock` on automatically if you would prefer.

Window Managers and Desktop Environments will also allow customization of the keyboard and mouse (as long as it is recognized correctly by X). This may be an easier way to configure certain customizations.

4.2.1. Special Key Mappings

There are several special key mappings traditionally used in XFree86.

Ctrl-Alt-BackSpace
: Will kill the X server process in an orderly fashion. This is a quick, easy, legitimate way to restart X. Note it does not restart the display manager (if used) — just X itself.

Ctrl-Alt-F*n*
: where *n* corresponds to a valid TTY number (typically 1-6). This is typically used to jump to a text console login, while X remains running. To get back to X, press **Alt-F*n***. In this case, *n* represents one plus the last TTY (e.g. **Alt-F7** if there are six available TTY's).

Ctrl-Alt-+ and **Ctrl-Alt--**
: That is the plus and minus keys on the keypad. This will cycle through any existing valid screen resolution modes, e.g. 1024×768   600×800. Note the actual screen size is the same — just the view and resolution changes. Not all that useful for most purposes. You cannot permanently change the screen resolution without restarting X.

It's possible your Window Manager, Desktop Environment or other system component may trap these, and alter the standard behavior. In addition, the **Ctrl-Alt-Delete** may be trapped as well. This should shut X (and the system) down orderly, if it is available.

4.2.2. Mice and Pointers

As mentioned, Linux and Unix make heavy use of three mouse buttons. If a mouse only has two buttons, then the third (i.e. the middle) button can be simulated by pressing both buttons simultaneously. This is a configuration option set in `XF86Config` as the "Emulate3Buttons" directive:

```
Section "InputDevice"
       Identifier  "Mouse0"
       Driver      "mouse"
       Option      "Device" "/dev/mouse"
       Option      "Protocol" "PS/2"
       Option      "Emulate3Buttons" "on"
EndSection
```

When all is said and done, a third button is quite handy and I would personally recommend having one. On wheeled mice, the "wheel" acts as the third button, if pressed. Many standard wheel mice seem to work with the "IMPS/2" protocol option.

Specifically, the third button (middle) is the "paste" button in virtually all Linux applications. Copy and paste works a little different in Linux. The left button is the copy button. Just hold it down, and drag over text. It is automatically copied to the X "clipboard". Then, the middle button will paste from there. A very simple process. A double-click should copy individual words, and a triple-click individual lines of text. If for some reason, this does not work, it is either a poorly implemented application, or a bug of some kind. Some older versions of Netscape were not consistent about this, for instance. To paste from the keyboard, this should be shift+insert.

"Drag and Drop" is not natively supported by X itself. But, is implemented by some toolkits and Desktop Environments. One should not expect this to work with non-compliant applications (i.e non-KDE aware applications in KDE for example).

4.3. xset

xset is yet another XFree86 utility to set user preferences. **xset** is a bit of a catch-all and is used to change various, unrelated X server settings. Mostly this is a command line way of configuring some of the same things that are defined in `XF86Config` (but not everything!).

Common usages of **xset** are to set DPMS on or off and preferred intervals, to dynamically change the FontPath or re-read it, to control keyboard LEDs, to adjust mouse (or other pointer) movement speed, set keyboard "autorepeat" and "repeat" rates, and to control X's built in screen blanking. See the man page, of course, for detailed explanations, and other **xset** usages.

Again, **xset** can be used interactively from the command line. But most often preferred settings are stored in one of the start up configuration files, like `.xinitrc` or `.xsession`. A very brief example:

```
# Turn off screen blanking
xset s off

# Enable DPMS energy saving
xset +dpms

# Tweak the rodent
xset m 30/10 4

# Speed up keyboard
xset r rate 200 40
```

Your desktop may have a GUI front-end for **xset**.

5. Fonts and Colors

Understanding fonts and colors can be more complex in X than on other platforms.

5.1. Fonts Demystified

X knows about various font types, including bitmaps, Type 1, and as of v4.x, TrueType. The X server can either handle fonts itself, or sometimes this duty is forked to a font server (of which there are several). **xfs** (X Font Server) is the most common font server in use on Linux.

A font server is not required, as X can handle most font rendering itself. Font servers are traditionally used for serving fonts to multiple hosts on a network, but sometimes are also used to provide enhanced functionality. Additionally, a font server may provide a modest performance boost by off-loading font rendering to a separate process.

X knows about fonts according to fonts that are in the "FontPath". This is set initially in `XF86Config`. If the X server is handling font duties itself (i.e. no font server), this will be a list of directories that contain font files, like:

```
FontPath        "/usr/X11R6/lib/X11/fonts/misc:unscaled"
FontPath        "/usr/X11R6/lib/X11/fonts/100dpi:unscaled"
FontPath        "/usr/X11R6/lib/X11/fonts/75dpi:unscaled"
FontPath        "/usr/X11R6/lib/X11/fonts/Type1"
FontPath        "/usr/X11R6/lib/X11/fonts/misc"
FontPath        "/usr/X11R6/lib/X11/fonts/100dpi"
FontPath        "/usr/X11R6/lib/X11/fonts/75dpi"
```

If a font server is being used, the "FontPath" will point to the socket where the font server is serving (this is just one possible example):

```
FontPath "unix/:7101"
```

In this latter case, the actual font directories that are available will be configured with the font server (see local documentation), which will use a similar directory type scheme as shown for `XF86Config`.

Once suitable fonts have been installed, they must be "prepared". For most fonts, this means running the **mkfontdir** utility (see man page) in the directory where the fonts are (as root). Type 1 and TrueType require additional steps (see below). Your vendor has done this for any fonts that were included with your distribution. So, this will only need to be done for fonts that you add. For newly added fonts to become visible to X, you will need to run the appropriate **xset** commands to either modify the existing FontPath, or re-read it (see man page). Or, re-initialize your font server.

Example: Preparing fonts, and re-initializing font server after adding new fonts:

```
su
<password>
mkfontdir /usr/X11R6/lib/X11/fonts/my_new_fonts/
/etc/init.d/xfs restart
```

The first command may not be necessary on newer distros (since it's done by the init script in some cases). And the font server configuration would need to be modified, if this is a new directory. Example: re-initializing with no font server:

```
su
<password>
mkfontdir /usr/X11R6/lib/X11/fonts/my_new_fonts/
xset +fp /usr/X11R6/lib/X11/fonts/my_new_fonts/
xset fp rehash
```

The "**xset +fp**" would not be necessary if the directory is already part of the FontPath.

xlsfonts | less can be used to list what fonts are known, and thus available, to X and its clients. Run **xlsfonts | less**, and you also can get an idea of the font definition as understood by X. Font resources are specified quite explicitly, and it may seem complex at first. The *X Logical Font Description* ("XLFD") is the full description for any given font. The XLFD looks like:

```
-adobe-helvetica-medium-r-normal-*-*-120-*-*-p-*-iso10646-1
```

Where each field, left to right is:

`fndry` - font foundry, the company or individual which made the font.

`fmly` - font family, the popular nickname of the font

`wght` - font weight (bold, medium, etc.)

`slant` - font slant (italics, oblique, roman (normal), etc.)

`sWdth` - font width (normal, condensed, extended, etc.)

`adstyl` - additional style (sans serif, serif, etc.)

`pxlsz` - pixel size, the number of pixels vertically in a character

`ptSz` - approximate point size of the text (similar to pxlsz)

`resx` - horizontal resolution, in dpi

`resy` - vertical resolution, in dpi

`spc` - spacing, only useful, apparently, in the Schumacher fonts

`avgWidth` - average character width of the font

`rgstry` - the recognized registry that lists the font `encdng` - nationality encoding

The "*" acts as a wild-card character. In fact, if not every field is specified, the X server will take the first match it finds in the FontPath. This is why it is best to order the FontPath with preferred fonts coming first since some programs will deliberately specify fonts "loosely" so that your system has some discretion.

The program **xfontsel** (X Font Selector) may be useful. Try launching it now. You will see nothing helpful in the main window at first, but try holding the left button down on the `fndry` button. If all your fonts are in order, you will see a menu of selections such as `adobe` and `b&h` and `bitstream` and so forth. Select one such as `b&h` and you will notice that the font in the lower window changes to something intelligible. This is the way fonts are selected with this program; starting from the left, which is the most general selection, and moving toward the right, to the more specific options. Selecting an option toward the rightmost end will not make much sense before the foundry, for instance, is selected, because the options are generally ordered by their dependence on each other.

When you select from the `fmly` selection, you will see most of the options grayed out, and only three remaining. That means that these three are the only families of font made by this foundry. Some families appear under more than one foundry, for instance, both *Adobe* and *Bitstream* make a variation of the Courier font. Now you can select the `wght`, and so forth. After you get far enough you will have narrowed it down to the font that you want. You don't necessarily have to fill in all the options to choose a single font, there's not *that* many fonts on your system! The options that you do not select will be represented by a `*` indicating that any option will do in that spot, and gives X some leeway.

When you are satisfied with your font selection, hit the select button, and your selection will be placed in the X clipboard, ready to be pasted into your document or whatever you are working on. For example, open an *xterm* window and type in something like `xterm -font` followed by an opening quotation mark. Then point to that spot on your screen, and click your middle mouse button (or click both the left and right, if you are middle-button impaired). This will paste the selection from the clipboard, which should be the font you just selected. Then enter the closing quote, and hit `Enter`. For instance, a nice big *xterm* with a Courier font specified would look like this: `xterm -font "-adobe-courier-medium-r-*-*-14-*-*-*-*-*-*-*"`.

If you've found a font you prefer, this can permanently be used by placing the font definition in the appropriate configuration file (see above).

Note that you can also limit the number of fonts that you want **xfontsel** to display with the command line option *-pattern*, followed by a quoted font specification, as discussed above.

The **xfd** utility is also helpful for examining individual fonts. If launched with a command line such as **xfd -fn fixed**, it will show you the complete character set for that font.

KDE and GNOME have their own utilities that are not quite as obtuse ;-)

5.1.1. Type 1 and TrueType Fonts

The fonts provided with XFree86 are of limited use for many of us, considering that about the only place you'll find fonts of that kind, are used in the X Window System itself for the most part. Unfortunately many media junkies, web designers and fontaholics work in operating systems that rely on other formats. And then, there often does not seem to be much emphasis by some distributions on making the best of the default fonts either.

Type 1 fonts, most commonly used in conjunction with PostScript document formats, are the traditional standard in Unix and Linux environments. You should have a reasonably good starter selection installed already. Or, more can be found for free on the Internet with considerable ease, and Try ftp://ftp.cdrom.com/pub/os2/fonts/ for starters. Type 1 are scalable fonts, and have many of the same benefits of the better known TrueType fonts. If you don't have a good selection of TrueType fonts installed, then Type 1 is what you want for most GUI applications. But again, this is not standard on other platforms, and can present problems when viewing documents (e.g. web pages) that are designed with "other platforms" in mind.

TrueType fonts started with Apple, and later were licensed by Microsoft. So people migrating from non-Unix platforms are already familiar with these high quality fonts. Unfortunately, there are not many quality TrueType fonts under a suitable license, and thus there are not many included with Linux distributions. And the ones that are, often are not as high quality. Also unfortunately, TrueType has become somewhat of a standard on the Web and in other venues, and not having good TrueType fonts can be a detriment. XFree86 also seems to render TrueType a little better than Type1.

That is the bad news. The good news is that any TrueType font included with any version of Windows, or any Windows applications, should work on Linux. Though you will have to take some additional steps to integrate them. This particularly helps web browsing where X's bitmapped fonts just don't scale well.

We shall not go into detail on installing and configuring these fonts here, as it is addressed in depth in other documents. See The Font HOWTO for general font information, and Type 1 tips. See *The Font De-Uglification Mini HOWTO*, The Font De-Uglification Mini HOWTO, for various X related font tips, especially TrueType.

5.2. Colors

Let's go back to our terminal window and try something. Open an **xterm** with a command line like the following:

```
xterm -fg DarkSteelBlue1 -bg red3 &
```

Ouch! While that may not be pretty, and you may not do much of your best work in it, it demonstrates one interesting aspect of X configuration -- color names. While not particularly precise, this is a nice way to remember a variety of colors. Note that color names are never case-sensitive.

The X server will actually deal with color values as a hexadecimal Red-Green-Blue (RGB) color notation. This would look something like "#0aff0a" in hex. Not so easy to remember. But X gives a more mnemonic way of remembering valid color definitions. These are stored in a text table, typically as `/usr/X11R6/lib/X11/rgb.txt`, and is defined in `XF86Config` in the "Files" section.

If you are interested, have a look with a text editor. There are many, many shades defined. I count eighty-three shades of blue in mine, for instance. Brief snip:

```
176 226 255             LightSkyBlue1
164 211 238             LightSkyBlue2
141 182 205             LightSkyBlue3
 96 123 139             LightSkyBlue4
202 225 255             LightSteelBlue1
188 210 238             LightSteelBlue2
162 181 205             LightSteelBlue3
110 123 139             LightSteelBlue4
```

```
191 239 255             LightBlue1
178 223 238             LightBlue2
154 192 205             LightBlue3
104 131 139             LightBlue4
```

This file can be customized should you desire, but this is rarely needed for most of us. It is important to have though, since some applications depend on it.

Desktop Environments will have a GUI utility for selecting colors.

6. Window Managers and Desktops

We shall not delve into configuring Window Manager's and Desktop Environments. There is just too much to try to cover in one document. It is important to realize that the two are not the same. There are many, many Window Managers available.

6.1. Window Managers

Window Managers are highly configurable. Many aspects of user interaction can be controlled by the Window Manager.

Some of the most popular Window Managers:

aewm: http://www.red-bean.com/~decklin/aewm/

AfterStep: http://www.afterstep.org/

BlackBox: http://sourceforge.net/projects/blackboxwm

Enlightenment: http://www.enlightenment.org/pages/main.html

Fluxbox: http://fluxbox.sourceforge.net/

fvwm: http://www.fvwm.org/

IceWM: http://www.icewm.org/

olwm (OpenLook Window Manager): http://www.plig.org/xwinman/olvwm.html

Sawmill: http://sawmill.sourceforge.net/

WindowMaker: http://www.windowmaker.org/

XFce: http://xfce.org/

There are many, many lesser known ones as well. http://www.plig.org/xwinman/ has an updated list of Window Managers, and related information. There is always freshmeat too.

GNOME and KDE both have their default Window Manager, but support other, compliant Window Managers as well. Your distribution probably has included at least several. Try them all if you don't already have a favorite. Your distribution probably also has a method of switching dynamically between Window Managers (and Desktop Environments too).

6.2. Desktop Environments

Desktop Environments are not really new, but their popularity has increased with advent of the two big names: KDE and GNOME. To a certain extent, the Desktop Environment functionality overlaps the Window Manager's. They both can be responsible for the root window background, root window menu, icons, taskbars, etc. Generally speaking, if a Desktop Environment is running, it is controlling these aspects. That is the main idea behind them -- to integrate the various components into a cohesive, consistent whole. Desktop Environments also add some interoperability and ease-of-use features that a simple Window Manager cannot.

Oh, another point: Desktop Environments also try to do as much X session configuration as possible. Any of their compliant clients will more than likely be configured by the Desktop, or have it's own configuration that conforms to the Desktop's style. This is at least partly to avoid much of the seemingly helter-skelter text file configuration we looked at in the above sections, and make life a little easier for the user.

There is a trade-off in this additional functionality, and that is that it takes memory and system resources to oversee all this. If you have plenty of memory and a fast computer, this is no problem. But in low memory situations, this can cause a slowdown (see the performance section below). 64M of RAM is probably borderline with either KDE or GNOME.

So do you need a Desktop Environment? That is up to the user. They are certainly not required to run X, but do add features that many users want or expect in a GUI. Which one is better? Ah, but that is up to you to decide!

KDE has been around longer than GNOME, and some would say maybe a little more mature. KDE is based on the QT widget toolkit. A quote from the KDE home page:

> KDE is a powerful Open Source graphical desktop environment for Unix workstations. It combines ease of use, contemporary functionality, and outstanding graphical design with the technological superiority of the Unix operating system.
>
> KDE is a mature desktop suite providing a solid basis to an ever growing number of applications for Unix workstations. KDE has developed a high quality development framework for Unix, which allows for the rapid and efficient creation of applications.

GNOME is based on the GTK+ toolkit. And a quote from the GNOME home page:

> GNOME stands for GNU Network Object Model Environment. The GNOME project intends to build a complete, user-friendly desktop based entirely on free software. GNOME is part of the GNU project, and GNOME is part of the Open Source(tm) movement. The desktop will consist of small utilities and larger applications which share a consistent look and feel. GNOME uses GTK+ as the GUI toolkit for all GNOME-compliant applications.

XFce is a lighter weight, less featureful Desktop Environment that does not get as much attention as the others. XFce is also based on the GTK+ toolkit. And a quote from the XFce home page:

> The XFce project was first started because I needed a simple, light and efficient environment for my Linux System.

> I believe that the desktop environment should be made to increase user productivity. Therefore, the goal is keep most system resources for the applications, and not to consume all memory and CPU usage with the desktop environment.

All these have their own extensive documentation. If you can't find what you need installed on your system, check the respective home pages.

7. X and the Command Line

What would a Unix-like operating system be without a command line interface? The command line can be useful, and is readily available with X. In fact, for many it is an integral part of their X working environment.

Any X program can be started directly from the command line just by typing the program name at a shell prompt in an **xterm**, or other terminal window. Most applications will have a very rich set of command line "options", such as background color, font, geometry (screen placement), etc, etc. Command line options over-ride compiled in defaults, or other system enabled "resources".

Many traditional X programs will use the same basic names for command line options. All applications written using the MIT X Toolkit Intrinsics (Xt) (such as those included with XFree86) automatically accept the following options. Some non-Xt applications also use these, or something similar. For instance, "geometry" is close to a universally accepted option.

-display *[host]:display[.screen]*
: This option specifies the X server display to use. This is often used where applications are run on one system, and displayed on another. The application needs to know *where* to display. This is sometimes also accomplished by setting the "$DISPLAY" variable, which uses the same syntax.

-geometry *geometry*
: The initial size and location of the window, in a format such as `width x height +horz_offset +vert_offset` or `+horz_offset -vert_offset`. Note that if you put in a negative horizontal or vertical offset, the window will be placed counting backward from the right or the bottom of the screen, respectively, instead of from the top left corner.

-font *fontname*
: The font to use for displaying the text in your window (see font section below).

-bg *color*
: The color to use for the window background. Typically this is a "color name" (see below).

-fg *color*
: The color to use for the window foreground (i.e. fonts, etc).

-name *resource-name*
: Useful for specifying the name under which the resources for this application will be found (e.g. as specified in `.Xdefaults`). This is useful to distinguish between invocations of the same application. For example, two **xterms** can be "named" differently so that they may inherit different resources based upon the specified names in the resource database.

-title *string*
: This is the title to be used for the window on your display, generally used by the Window Manager to put a descriptive title at the top of the window. Not to be confused with the "-name" option.

-iconic
: Open window in an iconified state.

-xrm *resource-string*
: This option specifies a resource name and value to override any defaults that may already be set (i.e. via `.Xresources` or similar). Also useful for setting X resources that do not have explicit command line options. For example, the command line "xterm -xrm xterm*background: blue &" is functionally the same as "xterm -bg blue &".

These are the most noteworthy. There are others. Many programs will have their own additional options that are application specific. Many newer applications today don't necessarily adhere to the Xt standards, and will use their own options, or those provided by their respective toolkit. If nothing else, man pages are a good reference for command syntax, and are your friends here. Or, the application will have a "--usage" or "--help" command line switch to list available options:

```
$ gnome-terminal --usage
Usage: gnome-terminal [-?] [--disable-sound] [--enable-sound]
  [--espeaker=HOSTNAME:PORT] [--version] [--usage] [--gdk-debug=FLAGS]
  [--gdk-no-debug=FLAGS] [--display=DISPLAY] [--sync] [--no-xshm]
  [--name=NAME] [--class=CLASS] [--gxid_host=HOST] [--gxid_port=PORT]
  [--xim-preedit=STYLE] [--xim-status=STYLE] [--gtk-debug=FLAGS]
  [--gtk-no-debug=FLAGS] [--g-fatal-warnings] [--gtk-module=MODULE]
  [--disable-crash-dialog] [--sm-client-id=ID] [--sm-config-prefix=PREFIX]
  [--sm-disable] [--tclass=TCLASS] [--font=FONT] [--nologin] [--login]
  [--geometry=GEOMETRY] [-e COMMAND] [-x COMMAND] [--foreground=COLOR]
  [--background=COLOR] [--solid] [--pixmap=PIXMAP] [--bgscroll]
  [--bgnoscroll] [--shaded] [--noshaded] [--transparent] [--utmp]
  [--noutmp] [--wtmp] [--nowtmp] [--lastlog] [--nolastlog] [-t TITLE]
  [--icon=ICON] [--termname=TERMNAME] [--start-factory-server]
  [--use-factory]
```

7.1. xterm and friends

Sooner or later, most of us need to access the "command line" for one reason or another. For some, this might even be a common way of working in X. In addition to being able to launch X applications from the command prompt, there is also a wealth of programs that run in "text mode" for Linux.

This is possible via "terminal emulators" such as **xterm**. The closest counterpart from Microsoft is the so-called DOS-box, which is child's play by comparison. Linux terminals support color, full mouse copy/paste (and some wheeled mice), pseudo-transparency and pixmap backgrounds, scrollbars, menus and generally a slew of other features. While **xterm** is the best known such terminal emulator, there are many similar programs. To name a few: **Eterm**, **rxvt**, **aterm**, **konsole** (KDE) and **gnome-terminal**.

In typical usage, when a terminal emulator window is opened, a shell is started for the user to interact with. The default for essentially all Linuxes, is the bash shell. So when all is said and done, the user is interacting with X, the terminal, and the shell all at once. Each may have it's own influence. For example, how keystrokes are handled since they move from hardware to X server to terminal to the shell and finally echoed back to the user.

Quick and easy terminal configuration is done via the "$TERM" variable, which is typically set in one of the user's shell configuration files. Or the terminal itself will have a compiled in default. The default value for this is most often "xterm":

```
$ echo $TERM
xterm
```

Normally this is sufficient, as your vendor has already set this up in a reasonable way. The "$TERM" variable is actually a reference to an entry in the "termcap" database (man termcap), which is typically installed as `/etc/termcap`. Unless you are doing something really unusual, you probably will not need to change this. Some additional terminal configuration can be done with the **stty** command (see man page). Terminal configuration is really beyond the scope of this document.

The terminal application itself (e.g. **xterm**) will also have various configuration options. Permanent settings are best stored in a `~/.Xdefaults` or similar file for those applications that support this. Generally speaking, applications with a GUI configuration (such as **gnome-terminal**), will be configured by their own menu driven configuration instead.

Also, you are interacting with the shell too, which can have it's own impact, particularly on how keystrokes are handled at the shell prompt. For bash, this can be adjusted in `~/.inputrc`. Again, this is beyond the scope of this document, but check with either local or on-line bash (or other shell) references.

Terminal emulators like **xterm** require a monospaced font. So forget about TrueType or Type 1 fonts.

---

8. X Networking and Security

As mentioned, X is essentially a networking protocol with graphical displaying capabilities. This makes for some interesting usage possibilities. And also means there are inherent security considerations, as there is with any networking environment. And if you ever connect to the Internet, you are in the midst of one very large, hostile network ;-)

X clients connect to X servers via various networking protocols, including TCP/IP. Even with just local connections. Possible usages here are to run an application on one computer, and display it on another. Or, to actually log in to a remote system, and have it display to your local screen, with the client apps using the remote system's CPU and RAM.

Without any precautions, this can leave you wide open to various types of mischief and abuse. For instance, anyone logged into to your system can access your "display", meaning they can see what you are doing if they want to. Thankfully, most recent Linux releases come with some default security precautions enabled. But it is best to make sure for yourself that you are protected.

Both X networking and security are nicely covered in The Remote X Apps Mini HOWTO , so we shall not need to try to rehash it here. Recommended reading. See other references in the Links section of the Appendix below.

A few recommended precautions:

- Never, ever run X as root. The number of bad things that can happen, dramatically increases when logged in as root. Learn to run as much as possible as a regular user, and su to root only when needed. This may sound like a lot of extra work (and probably is at first), but once the "right" way of doing things is learned, it soon becomes second nature.

  A brief anecdote from a friend: he had a client who's new system stopped "working". Curiously, he found the entire `/dev` directory was missing, which he re-installed and all was well again. He was back a few days later and found the system logged in as root to X, and someone had clicked on `/dev` in the file manager, and dragged it onto the desktop. Smooth move!
- If you ever connect to a network with untrusted users, be sure to have a firewall between you and them. This goes double for the Internet. Firewalling is beyond the scope of this document, but is covered in many other places, including your vendor's website. http://linuxdoc.org has several security HOWTOs that can help as well. http://linuxsecurity.com/docs/ is another good place to look.
- You can disable TCP connections with the "-nolisten tcp" command line X server switch. This does not help for local connections though. For **xinit/startx**:

  ```
  exec X :0 -dpi 100 -nolisten tcp
  ```

  Placed in `~/.xserverrc`. And for **xdm**, in `/usr/lib/X11/xdm/Xservers`:

  ```
  :0 local /usr/X11R6/bin/X :0 -nolisten tcp
  ```

9. Performance Considerations

As has been discussed, what we call X, is actually a convergence of various components: X server, Window Manager, Desktop, etc. With MS Windows, the GUI desktop is tightly integrated with the operating system itself. This is not the case in Linux which follows the Unix tradition of combining various independent components to achieve some end result. So we have choices with each component and it's attendant configuration and implementation. In short, much flexibility. This is where you come in. You can try various possibilities and decide what you gives you the most bang for the buck.

On low end hardware, this gives us much latitude to decrease the demand on available system resources. This is good because, if given the opportunity, X can be quite greedy with system resources. If you've recently installed a new Linux distribution, you've probably been given a default Desktop with many bells and whistles. And something that will probably need a fair amount of memory and CPU to achieve a reasonable level of performance. If you have the horse power, this should not be a problem.

It is often said that Linux functions very well with relatively little memory. This is true to a point. It does not mean though that every possible configuration will run with low memory. So if you want to use memory hungry applications, then you will have to have the memory. Or you will have to make sacrifices to achieve a satisfactory level of performance. It is quite possible to run X with reasonable performance on 16 Meg of RAM, and even less if you really want to push it. But you would have to live with some real limitations.

Let's look at some of the components and ways to decrease the demand on system resources, in case you are at the low end on hardware, or performance is not up to expectations.

9.1. Hardware

- No big surprise, but overall system performance will be best with a fast graphics card, a fast hard drive, and lots and lots of memory -- if you want *both* a fast and flashy system.
- Graphics cards are of course necessary, and the X server's video performance is tied to the card's chipset, and the corresponding XFree86 driver. Just because a given card is supported by XFree86 does not necessarily mean it is as well optimized as other cards! It may also perform better at a lower color depth (see below). It may well be worth the trip to xfree86.org to see if there are any notes related to your card with respect to performance or other issues.
- And you might try other versions of XFree86. At this time v4.2 was just recently released. Some cards may still perform better with 3.3.6 due to the way v4.x is being incrementally developed. If you are using x4.x and performance is not good, then make sure you are using the latest available version.

9.2. Memory

The more memory, the better. X will do a lot caching to help performance. But caching requires memory, and if there isn't much to start with, then we would need to reduce memory requirements. Some tips for those with low memory or performance problems:

- Use the **free** command to make sure all memory and swap is recognized.
- Make sure you don't have other system services that are hogging memory or CPU. Use **top** or **ps** to see what is running, and disable anything you can to free up memory and CPU. Again, a default installation may have many things running that you don't really need.
- Make sure you have plenty of swap space. With low, or even modest, memory, swap is all the more

important. A general rule of thumb is twice as much swap as physical memory. With low memory, this is not enough. Try four times real memory. Or more. If you can't create more swap partitions, see the **mkswap** man page about creating swap files instead. Constant disk churning is a symptom of insufficient swap space, and the system will be slowed as a result, sometimes drastically. Or, possibly this may be the symptom of a poorly behaved kernel VM system (try another kernel in this case).

- Drive performance is important for swap performance. Make sure your drive has DMA enabled if the drive supports it, and is otherwise tuned and performing up to snuff. See the **hdparm** man page. Slow drive + slow card + low memory = slow system.
- Don't use KDE or GNOME if memory is tight. These both require substantial memory, and are not required to just run X. Think of these as usability enhancements. 32M probably may not be enough. 64M may be decent, depending on what other applications are being used, and other variables. 128M *should* be adequate in most situations. 256M or more to be comfortable. File Managers like Nautilus and gmc can also be memory hungry.
- Use a lightweight window manger. WindowMaker, BlackBox, IceWM, fvwm (and variants), XFce, all have reputations of performing well with low memory. There are surely others as well. Experiment. fvwm is generally considered the lightest of the light.

  A very nice desktop is still very possible even without KDE or GNOME. In fact, most KDE and GNOME applications can still be used even if KDE and GNOME are not running themselves (assuming the right libs are installed).
- Don't use fancy themes or backgrounds. Plain and simple is easier on resources. Use a solid color background. Avoid pixmaps or gradients for any kind of background, including menus, title bars, etc.
- Use a lesser screen size and color depth. 800x600x16 will not push X as hard and be easier on system resources than higher values. While a ColorDepth of 24 is preferred, you probably will not notice the difference of 16 with the majority of applications.
- Some applications require much more memory than others. Some notable hogs are **Netscape**, **Mozilla**, office suites, and **the Gimp**. **Netscape** is faster than **Mozilla** (but not as nice). **Netscape-Navigator** uses less memory than **Netscape-Communicator**. Close any of these apps when not in use. Use text browsers like **lynx** or **w3m** wherever you can, like reading locally installed HTML documentation. Much faster, and much less memory is required.
- Also, use text based clients for mail (**mutt** or **pine**) and news (**slrn** or **trn**). Again, faster and much less memory is used, and these are after all text based protocols at heart anyway.
- **rxvt** uses less memory than **xterm**, **konsole** or **gnome-terminal**.
- If you run an X session for long periods of time (like days or weeks), restart X occasionally to free memory tied up as cache.
- Disable "backing store" and "save-unders" to reduce memory usage (performance penalty though). Check your Window Manager's settings too. See what modules are being loaded in the "Modules" section of `XF86Config` as well. Your installation may have many unnecessary ones enabled, or ones you can't take advantage of (e.g. "v4l", aka "Video4Linux").
- Font servers may provide a slight performance boost by off-loading font rendering to the font server, while freeing the X server to do other things. But, the font server will use a small additional amount of memory as well. So, you can try it either way to see if it makes a difference.
- Lastly, RAM is cheap now. Buy some ;-) A new drive too.
- RAM is still just too low for X? Check out tinyX: http://www.superant.com/smalllinux/tinyX01.html. Reportedly runs in as little as 4 Meg of RAM.

9.3. X over the Network

X is not particularly network friendly. In other words, it is a bandwidth hog. This should not be a problem in LAN situations, but may be if trying to use X over the Internet.

- **lbxproxy**, the low bandwidth X proxy, utilizes various optimizations to improve performance in low bandwidth, or high latency situations. See the man page.
- VNC (Virtual Network Computing), has some of the same advantages as X for displaying applications on remote systems, but is more network friendly. Your Linux installation should have both VNC client and server packages available.

9.4. Other Tips

Other tips to eek out better performance:

- Use **xset** to speed up the keyboard. This can make the system feel more responsive even if it isn't really. The default always seemed sluggish to me.
- **renice** X to give it a higher priority. Other platforms give the GUI high scheduling priority to achieve better responsiveness. But this is at a cost to other processes. Linux is a blank slate. You might include the font server (if being used), and key KDE and GNOME processes as well.

  ```
  nice -n -10 X :0
  ```

  This will not do much on systems that are mostly idle.

  This does not work so well with **startx** since X runs as root, and you are not root, right? So you would have to use something like **sudo** to have this done automatically.

10. Appendix

10.1. Terminology and Usage

There are a few basic concepts and terminologies you should be familiar with. These terms will appear here, in the manual pages, and in other help files and documentation.

- The "X server" is the low-level driver software that interacts with your video card and other system hardware, and manages the "display" and the various components attached to the "display" (keyboard, mouse, etc.). And, of course, handles requests from clients as well. There are different X servers for different chipsets.

  X Servers are referenced in the form of:

  host.domain:display_number.screen_number

  An example would look like: *my_computer:0.0*

  If host (and domain) is omitted, localhost is assumed. "Host" can be a remote host. If "screen" is omitted, then "0" (the first screen) is assumed. In it's shortest form, the X server is often represented as just ":0", which would be the first local "display". X supports multiple "displays" *and* multiple "screens".

  "Screen" and "Display" have special meanings in relation to X servers, in addition to their more common usage.
- When X is invoked, the X server will initialize one or more "displays". Yes, X can have more than one "display" available (though this is not a common configuration for the average user). Each "display" is a separate instance of "X". The "display" includes not only the obvious video components, but also the keyboard, mouse and other input type components. The user can only access one display at a time via the same keyboard and monitor. "Displays" may reside locally, or on a networked host "somewhere", or both. It is possible that if multiple "displays" are available, the user can choose which one he wants when he logs in. Each "display" may have its own unique configuration (e.g. resolution). But again, the most typical configuration is just one "display" with one "screen", which is how most of us use X.
- In reference to X servers, "screen" means the primary video output with which you view X. And there can be more than one "screen", just like you can have more than one "display". Additional "screens" are used in "multi-headed displays" for instance. In fact you can even have more than one computer running off a single X server. This is beyond the scope of this document, but you should be aware of this degree of flexibility as it is an important ingredient of the X protocol.
- "Desktop" can mean different things in different contexts. Often, "desktop" means what is more properly called the "Desktop Environment". Prime examples of this are KDE, GNOME, and the not as well-known CDE, which are high level applications that control much of how the user interacts with the X session. They provide consistent look and feel, as well as consistent configuration and come bundled with their own set of utilities for common tasks.

  "Desktop" also sometimes just means the viewable screen area. This is more of the MS Window's meaning. X environments though are capable of having multiple virtual "desktops" that can be switched between as needed. This helps with organizing different tasks. Each "desktop" may its own windows and clients that are specific to it. Right now I have seven WindowMaker desktops

(WindowMaker calls them "WorkSpaces"), and one of those I have dedicated to writing this document. This "desktop" has thirteen unique windows at the moment (man pages, browser windows, clock, gvim, xterms, etc).

- "Clients" are any program that connect to the X server, and require an X server for some task (e.g. to display itself). Often, these are displayed in their own "window", but not always. For instance, if I use `CTRL-N` to open a new Mozilla window, this is one X client but with two windows. If I run a command line X utility like **xev** to view key and mouse events, this runs in the **xterm's** window, so has none of its own, but is still a "client". Clients can be locally running applications, or applications that are running on another system over the network, but are displayed locally.
- The "Window Manager" is a special type of client application and a user definable component of the GUI. It is what the user interacts with to a large extent. The Window Manager provides such functionality as window borders and decorations, menus, icons, virtual desktops, button bars, tool bars, and allows the user to customize these. It is technically possible to run X without a window manager (though not very functional), but not the other way around. Window managers should not be confused with "Desktop Environments" like KDE. Desktop Environments include their own preferred Window Manager, but this is a configurable. There is some overlapping of responsibilities between Window Managers and Desktop Environments.
- The "root window" is the background of your screen. It is referred to as a window in name alone, it does not behave like any other window, but rather you run your applications on the root window, or put an image on it, or perhaps just a solid color. All other windows are children of this parent window. The root window conceivably can be larger than the viewable screen area.
- The "pointer" is the arrow or indicator of any given shape which represents the location of your mouse, or other pointing device. The pointer often changes to give you contextual feedback as to what will happen when you use the mouse at that point on the screen.
- The "window" is a frame in which any given application runs and which is "managed" by the Window Manager. This includes pretty much anything except the so-called root window. Even windows which do not appear to have frames, titles, or normal borders of any kind are being managed by your window manger. The "active window" is the window you are currently using. This window will will respond to the keyboard when you type, and is traditionally denoted by the fact that your mouse cursor is pointing at it, though this is not always the case. The active window is said to have "focus". Most Window Managers will somehow highlight the "active", or focused, window to differentiate it from other windows.
- "Menus", "icons" and "task bars" behave in X similar to the way they behave in other windowing systems, and the same general principles apply.
- Windows that run text only applications are called "terminal emulators", such as **xterm** and various similar applications. This is the well-known "command line" in an X environment. These basically emulate a console text-only display, and have some advantages due to their being used in X. These are much more complex and sophisticated applications than a simple DOS box on Windows.
- "Widgets" is the term used to describe such GUI control components as buttons, sliders, menus, scrollbars, listboxes, checkboxes, etc. "Toolkits" are libraries containing a diverse set of widgets with the same look and feel. Some common examples are GTK+ (used by GNOME, Mozilla and others), Xaw (X Athena Widget set), Tk, Motif and QT (used by KDE). Applications are built with one toolkit or another. Sometimes the same application can be built with different toolkits, depending on compile time options.
- Window "geometry" is a shorthand way of expressing a window's size and screen placement. This might look like "60x20+10+50", which is `WIDTH x HEIGHT +VERT_OFFSET +HORZ_OFFSET`. While both pairs are often specified, it is permissible to use just one or the other pair.
- In X lingo, "resources" are definable application attributes. Commonly available "resources" are fonts, colors, size, window title, etc, etc.

10.2. Links and other References

- The definitive source of information on XFree86 is, of course, http://xfree86.org. Don't forget the man pages that you have installed already too (X, Xserver, XF86Config, XFree86, xdm, xinit, xmodmap, startx, xauth, Xsecurity, etc, etc). These are really mostly decent, though some are quite technical.

  Some pages at xfree86.org to check:

  Docs and support info: http://www.xfree86.org/support.html for various versions and topics.

  README: http://www.xfree86.org/current/README.html

  Release Notes: http://www.xfree86.org/current/RELNOTES.html

  DRI: http://www.xfree86.org/current/DRI.html

  Status: http://www.xfree86.org/current/Status.html

  Mouse: http://www.xfree86.org/current/mouse.html

  Supported card list: http://xfree86.org/cardlist.html
- Other related documents from LDP:
  - If you are just starting out, you may find the X Window System Architecture Overview HOWTO to be helpful. It covers all the basic concepts quite well.
  - The Remote X Apps Mini HOWTO does a nice job of discussing running X remotely, and related security issues of X networking.
  - The XDMCP HOWTO covers the X Display Manager Control Protocol, for running X remotely. Also, The XDM and X Terminal mini-HOWTO .
  - The XFree86 HOWTO succinctly covers installation, and initial configuration.
  - The XFree86 Video Timings HOWTO gets down and dirty with the finer points of monitor tuning. Generally not required for XFree86 v4.x.
  - The Xinerama HOWTO covers multi-headed displays.
  - The Font HOWTO covers various font topics.
  - The Font De-Uglification Mini HOWTO covers a range of X font issues.
  - The International Keyboard HOWTO
  - The Linux Infrared HOWTO
- Looking for information on a Window Manager, or wanting to try something new or different: http://www.plig.org/xwinman/
- Wheel mice tips and configuration: http://koala.ilog.fr/colas/mouse-wheel-scroll/
- Linux and Laptops: http://www.linux-laptop.net/
- The O'Reilly series on X Window! Visit http://www.ora.com/ for the definitive books on X.
- The X Consortium's web site is http://www.x.org/ ... or perhaps it's moved to http://www.opengroup.org/.
- http://www.x11.org/ is sort of a clearinghouse for all things X.
- And for everything else under the Sun: http://google.com/linux/. An incredible resource in its own right.

Notes

[1]

```
v4.x changes:
- artheader tag changed to articleinfo
- graphic tag is being depricated in DocBook 5.x. To prepare for
  this, you should instead use the mediaobject tag.
- file format for imagedata has to be in capital letters.
- added support for PNG (notation in the DTD)



  --s+ [ser@metalab.unc.edu]

 aspell -H -c ~/ldp/x-user/LDP/howto/docbook/XWindow-User-HOWTO.sgml
 submit@linuxdoc.org

 export CVSROOT=:pserver:hal@cvs.linuxdoc.org:/cvsroot
 cvs -d $CVSROOT login
 pword: XXXXXXXXXXXXXX

 $cvs get LDP/howto/docbook/XWindow-User-HOWTO.sgml

 upload...
 $ cvs commit XWindow-User-HOWTO.sgml     !!!!!!!!!!!!!!!
 (from the LDP/howto/docbook/ dir)

check here: http://cvs.pld.org.pl/LDP/howto/docbook/XWindow-User-HOWTO.sgml

===================================
Start 3.1:
 TODO:

Changes:
    http://www.plig.org/xwinman (plus freshmeat)
    VNC vs X for performance on network.
    lbxproxy, performance.
    Add fluxbox.

Start rewrite 01/13/02 Hal Burgiss.

Changes:
 Converted to DocBook
 Rewrite! Rewrite!
 A zillion changes.

ToDo:
 $DISPLAY
 http://www.superant.com/cgi-bin/smalllinux.pl?smallX
 http://www.linuxgazette.com/issue27/kaszeta.html (xdm)
```

http://linux.daphnis.com/RedHat/Custom-X-Tips.html#toc4 (Xclients, etc)
http://www.linuxjournal.com/print.php?sid=5304 (X config, etc)
log file

http://www.linux.gr/cgi-bin/man2html/usr/X11R6/man/man7/X.7.gz
http://www.x-docs.org/

```
startx &> log

_X11TransSocketUNIXConnect: Can't connect: errno = 111 giving up.
xinit:  Connection refused (errno 111):  unable to connect to X server
xinit:  No such process (errno 3):  Server error.

## Makefile to build this doc as HTML and TXT ####################

# X Windows User HOWTO Makefile
#
# Hal Burgiss hal@foobox.net
#

TITLE = XWindow-User-HOWTO
EXT = sgml
SRC_DIR = LDP/howto/docbook
HTML_DIR = X-USER
SRC = $(SRC_DIR)/$(TITLE).$(EXT)

BUILD = jade -t sgml -ihtml -d /usr/lib/sgml/stylesheets/ldp.dsl\#html
BUILD_TXT = jade -t sgml -i html -d /usr/lib/sgml/stylesheets/ldp.dsl\#html -V nochunks
SPELL_CMD = aspell -H -c
LINKS_CMD = /usr/bin/linkchecker *html
EDIT_CMD = /usr/bin/vim -g
TBROWSER = w3m
TMP_TXT = __tmp.sgml
WWW=/var/www/html/ldp/x-user
SRC_URL = http://feenix.burgiss.net/ldp/x-user/$(TITLE).$(EXT).gz

all: doc txt

doc: html

html:
rm -fr $(HTML_DIR)
mkdir -p __tmp_htmls
mv -f *.html __tmp_htmls 2>/dev/null || :
mkdir -p $(HTML_DIR)
$(BUILD) $(SRC) ||\
 (rm -f *.html && mv -f __tmp_htmls/* . || : && rm -rf __tmp_htmls && false)
mv -f *html $(HTML_DIR)
mv -f __tmp_htmls/* .  2>/dev/null || :
rm -fr __tmp_htmls
```

```
clean:
rm -fr $(HTML_DIR) *~  __tmp_htmls

edit:
$(EDIT_CMD) $(SRC)


spell:
$(SPELL_CMD) $(SRC)

spellchecker: spell


links:
cd $(HTML_DIR) && $(LINKS_CMD)

linkchecker: links

linkcheck: links

txt:
$(BUILD_TXT) $(SRC) > $(TMP_TXT).html
$(TBROWSER) -dump $(TMP_TXT).html > $(TITLE).txt && gzip -f $(TITLE).txt && rm -f $(TMP_TXT).html

text: txt

www:
cp -fv $(HTML_DIR)/* $(TITLE).txt.gz $(SRC) $(WWW)
gzip -f $(WWW)/$(TITLE).sgml
rsync -auv $(WWW)/* feenix://$(WWW)/

submit:
@echo "Updated and ready:  $(SRC_URL)" |\
mail -s "$(TITLE) update" submit@linuxdoc.org &&\
echo " $(TITLE) Submitted!"
```