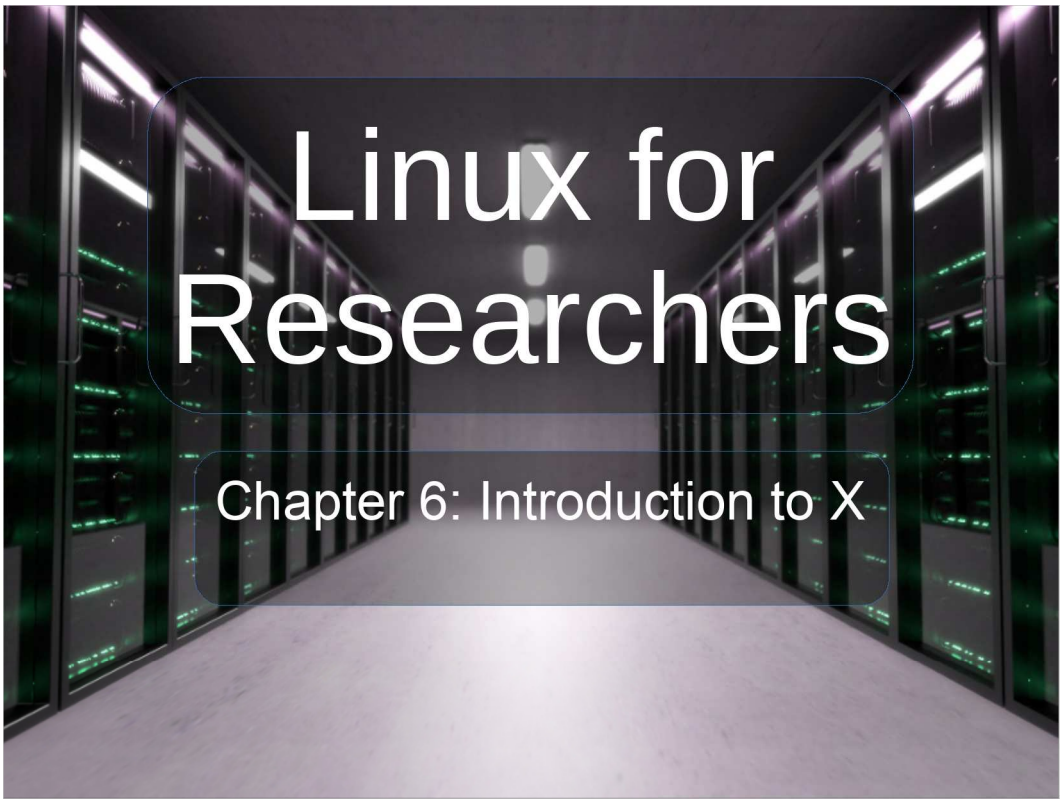


Up until now we've talked about command-line things. Now we'll take a look at a graphical interface.

There's no way we can thoroughly cover X in an hour, so what I'm going to say will just skim very lightly over the top of this enormous subject. I hope I can at least show you where the seams are, so you an go back later and pry it apart yourself, to look deeper inside.

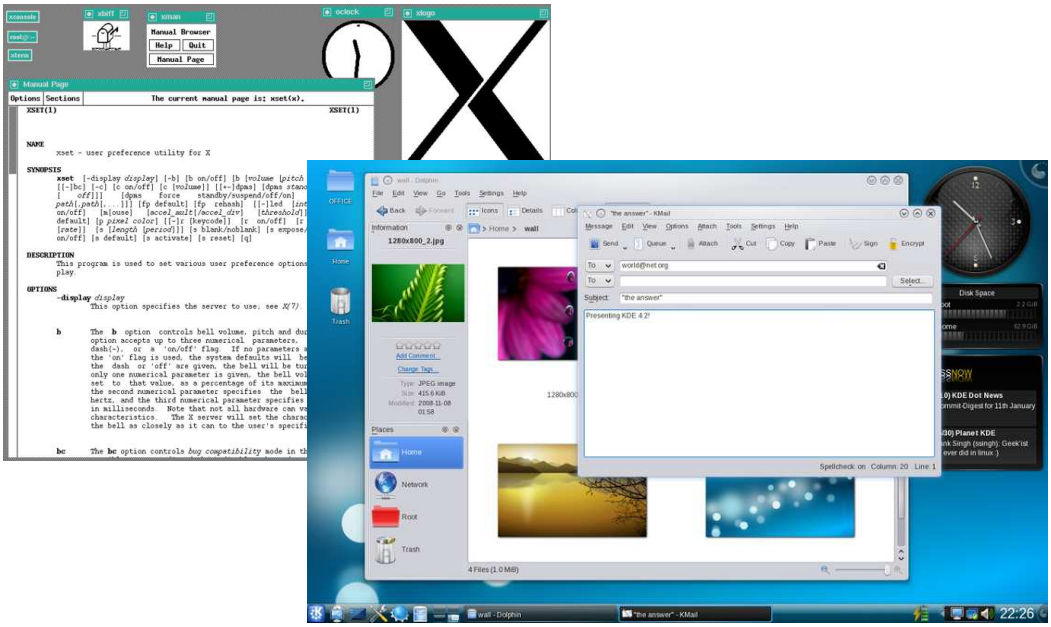
Part 1: The X Window System

We're all used to graphical interfaces now. Most people have never used a computer without one. But think back to those days of yore (say, 1980) when computers weren't graphical. Think about just how weird it is to point at what you want, instead of just telling the computer what to do. It's like being in a foreign country where you don't speak the language. All you can do is point to the visible choices. If something's available, but not visible, you can't select it.

Just like visiting a foreign country, a novice computer user doesn't know the language. The advantage of a graphical interface is that it lets the novice get work done right away, without any special training. Graphical interfaces made it possible for non-technical people to start using computers in fairly sophisticated ways.

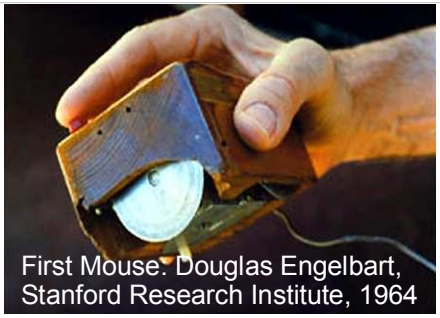
What is X?:

X is a OS-agnostic, hardware-agnostic, network-transparent, client-server system that provides a graphical user interface. It is highly configurable:



X is the most common, and most powerful, graphical interface used under Linux. X's configurability is great, but challenging. As we'll see, desktop environments like KDE and Gnome impose some order on what might otherwise be a daunting system, suitable only for computer geeks, and make it usable by novices.

The History of X:


First Mouse: Douglas Engelbart, Stanford Research Institute, 1964

- 1981, Xerox PARC develops the "Star" computer, with the first modern GUI.
- 1983, Apple releases the "Lisa" computer.
- 1984, X version 1 developed for the MIT "athena" project. Development began as a port of Stanford's "W" window system for the "V" operating system.
- 1984, Apple releases the first "Macintosh".
- 1985, Microsoft releases Windows 1.0.
- 1987, Current version (11) released. X development overseen by the "MIT X Consortium", a collaboration between MIT and industry (DEC, HP, Sun, etc.)
- 1992, Volunteers begin working on "XFree86", an X server for PC hardware, based on the MIT X distribution.
- 1992, Microsoft releases Windows 3.1.
- 1993, X Development taken over by "X Consortium, inc.", a non-profit vendor-controlled corporation. XFree86 continues to work in parallel.
- 1997, Taken over by "The Open Group", another vendor group. Attempted unsuccessfully to change X licensing. By this time, XFree86 is the most dynamic part of the X world.
- 2004, X development taken over by the "X.org Foundation", a community project open to (and governed by) individuals and funded by corporate sponsors.

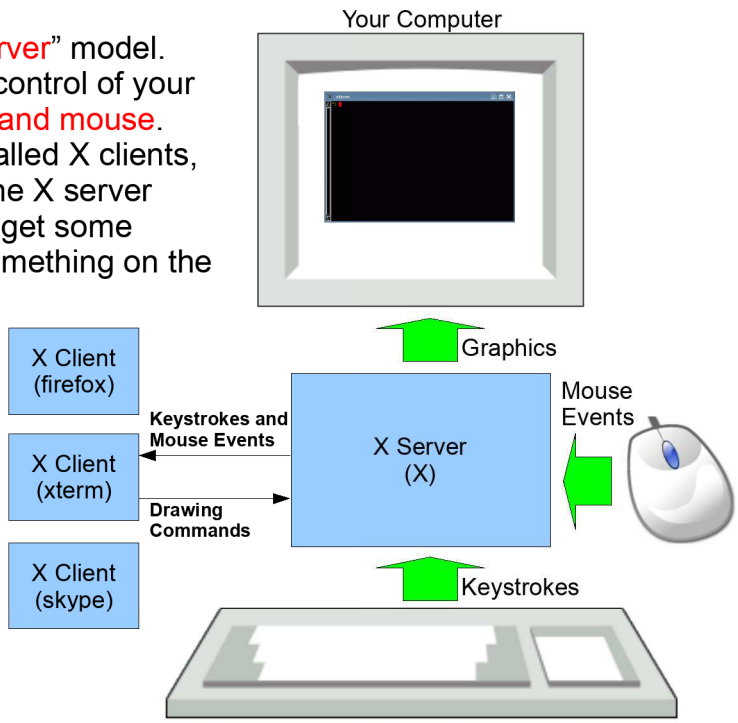
X development was almost dead for a long time during the 90s. In recent years, since the x.org foundation took over, development sped up considerably. The new community model, with bright, enthusiastic leaders like Keith Packard, brings frequent improvement and innovation.

Also note that "the mouse" was actually around for a long time before the folks at Xerox PARC started using it. Here's a nifty demo of this new device from 1968:

http://www.youtube.com/watch?v=1MPJZ6M52dI

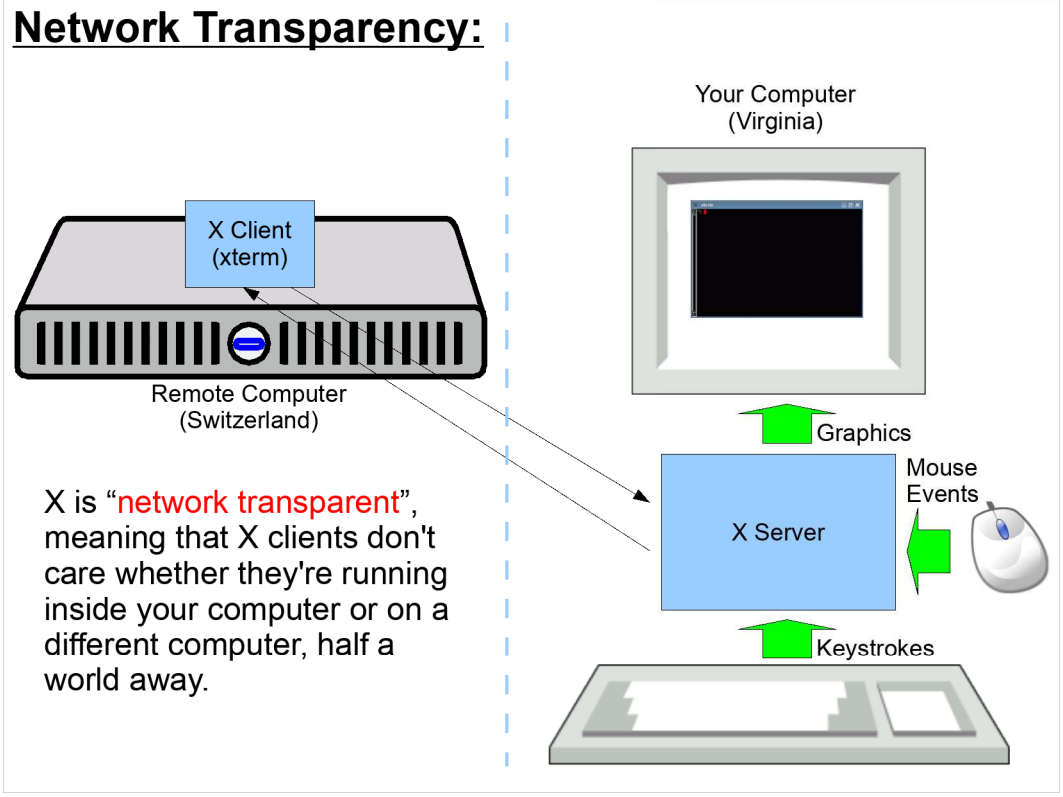
How X Works:

X uses a “client/server” model. The X Server has control of your monitor, keyboard and mouse. Other programs, called X clients, send requests to the X server when they want to get some input, or display something on the monitor.



In the example above, an X client program (xterm, in this case) is connected to a local X server. The server sends keyboard and mouse input to the client, and the client sends display commands to the server.

Note that the client doesn't need to care about what hardware the computer has. It just talks to the X server, and it's up to the X server to know how to draw on the display or get input from the mouse/keyboard.



X clients don't even have to be running on the same computer as the X server. Clients can talk to the server over a network connection. As far as the user is concerned, things look just the same as if the client program were running locally.

Advantages of X:

- Clients can run anywhere, and display anywhere. (Network transparency.)

- Clients don't need to care what type of video hardware your computer has. (The X server takes care of all that.)

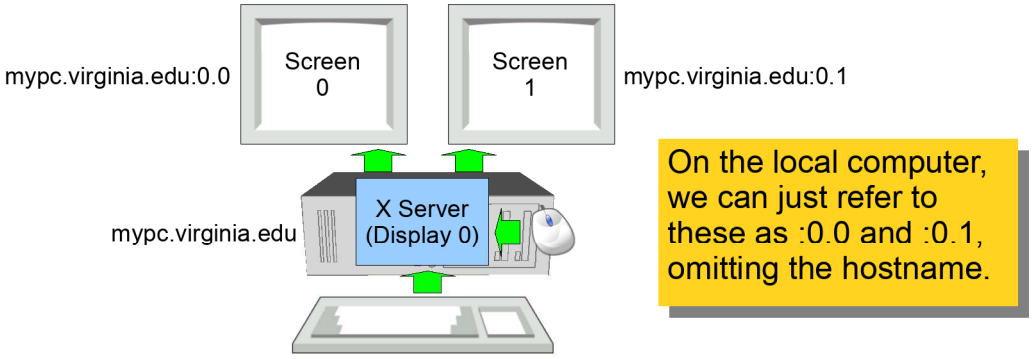
- Clients don't even need to be running on the same type of operating system. (All X clients use the same protocol, and can work together, no matter whether they're running under Linux, OS X or Windows.)

X Display Names:

To tell an X application which X server it should talk to, we use a "Display Name".

A computer can have multiple X servers running, and each X server process can control multiple physical monitors (“screens”). So, the Display Name has several parts:

```
hostname:displaynumber.screennumber
```

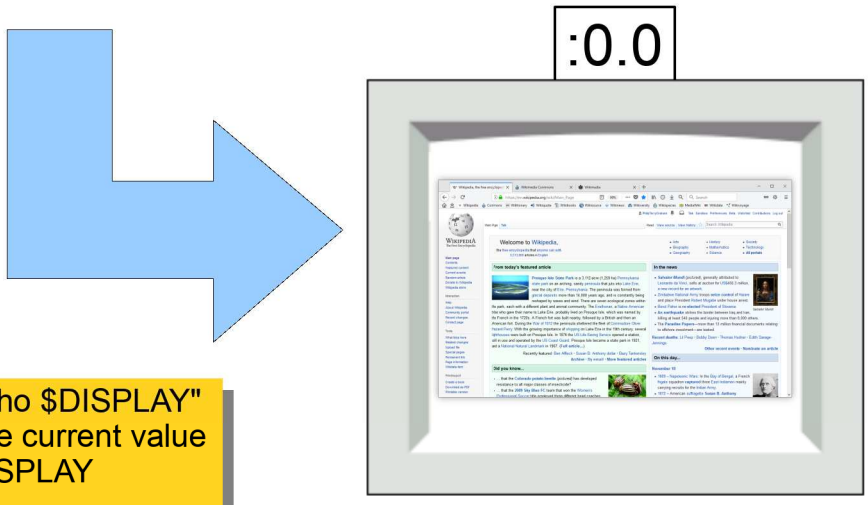


On the local computer, we can just refer to these as :0.0 and :0.1, omitting the hostname.

Later we'll see a good use for multiple X servers running on a single computer.

The "DISPLAY" Variable:

```
export DISPLAY=:0.0
firefox
```



Type "echo $DISPLAY" to see the current value of the DISPLAY variable.

The "DISPLAY" environment variable tells X applications which display they should use. Most applications will let you override this value by specifying the display on the command line, like:

```
xterm -d :0.0
```

Part 2: Running X:

The X server is just a program called “X”. There are many ways to start X, but it's usually started by a “display manager”.

X Display Managers:

The most commonly-used display managers are:
- gdm
- sddm
- lightdm

A Display Manager has several jobs:

- Start the X server, if it isn't already running.
- Connect to the X server as a client, and show a username/password dialog box on the X display.
- Wait for a user to successfully log in, and then start an X "session" (a predetermined set of X applications).
- Hang around silently until the session ends, and then re-display the username/password box.

There are many different display managers, but these are the most common:

Gdm. This is the Gnome display manager.

Sddm. This is the default display manager for KDE.

Lightdm. This is a highly-configurable display manager used by many distributions.

Starting the Display Manager:

• If you use **init**:
In /etc/inittab there will usually be a line like this:

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

The “respawn” flag in /etc/inittab tells init to re-start the display manager if, for whatever reason, it dies.

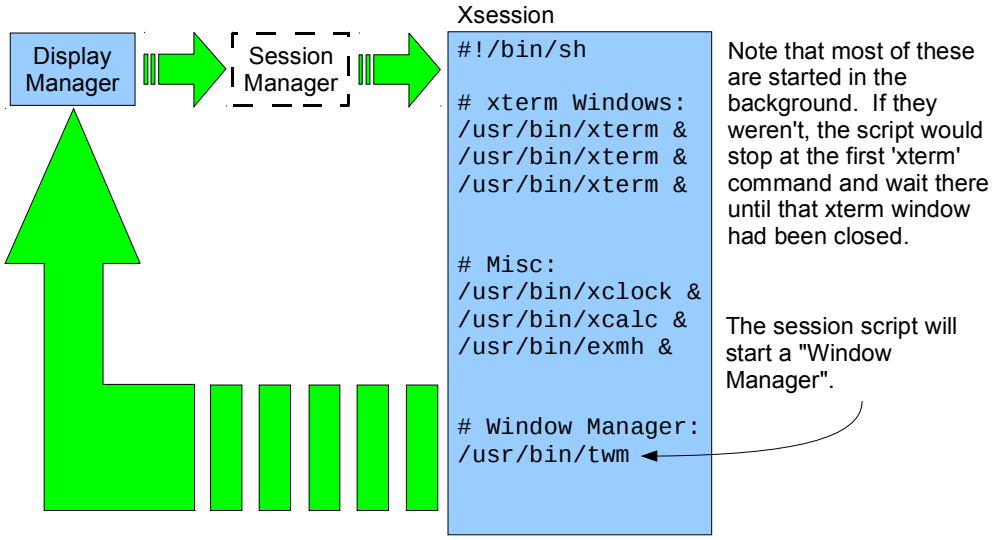
• If you use **systemd**:
You'll find a "unit" file like /usr/lib/systemd/system/lightdm.service which contains lines like:

```
[Service]
ExecStart=/usr/sbin/lightdm
Restart=always
```

along with other lines that describe the service and instruct systemd about other services it depends on.

X Sessions:

An X session is just a set of X clients that get started up when you log in. Sometimes, the display manager starts these by running a script (Xsession, xinitrc, etc.). Alternatively, the display manager might start a "session manager" to run this script.



Sometimes, there's one program in Xsession that acts like a stopper, keeping the script from exiting until the session is over. This can be done as above, by putting this application (twm here) at the end of the script, but there are other ways to do it, too.

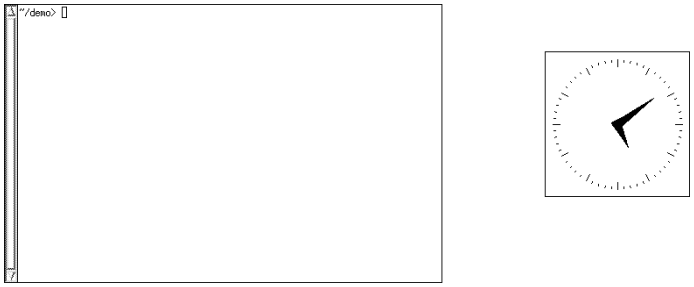
The Xsession script will typically be much more complicated than this example. Usually, it will be generalized so that the display manager can give the user a choice of desktop environments. The user's choice is then passed to Xsession as a command-line argument or environment variable.

By choosing a display manager and configuring its Xsession script, you can have very great control over how your computer's graphical environment behaves.

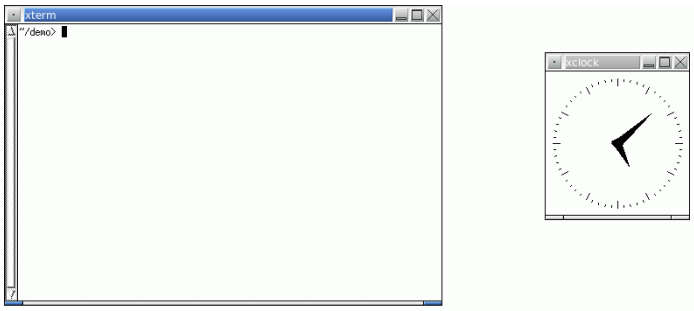
Window Managers:

A window manager is a special X client that has the ability to add decorations to other X clients, and to control them in various ways. For example, the window manager might let you move windows around on the display, resize or iconify windows.

Here's what an xterm and an xclock look like without a window manager....



...and here's what they look like when a window manager is running. Note the borders, banners and buttons.

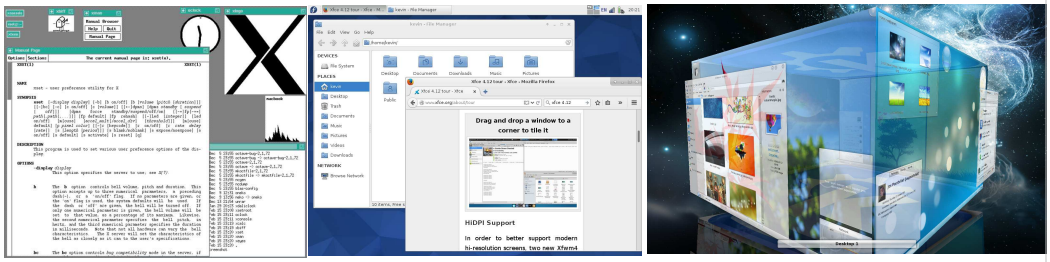


There are tons of windows managers. The one shown above is a lightweight window manager called “fluxbox”. One of the earliest window managers was “twm”, or “Tom's Window Manager”, written by Tom LaStrange in 1987. Modern integrated desktop suites like KDE and Gnome have their own window managers.

Window managers interact with other X clients through a set of standards known as the “Inter-Client Communication Conventions Manual”, or ICCM. Most modern window managers are ICCM-compliant, and so are to some degree drop-in replacements for each other.

What Window Managers Do:

- Decorate the windows displayed by other X clients, by adding banners, borders, buttons, etc.
- Allow the user to resize, move, iconify/deiconify windows.
- Provide the user with menus for configuring the window manager's behavior or performing other tasks.

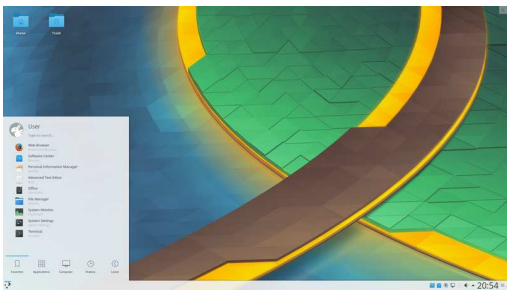


If we used "ps auxf" to look at the process tree, we might see something like this while we're logged into the computer:

```
lightdm
      \_ /bin/sh /etc/xdg/xfce4/xinitrc
          \_ xfce4-session
               \_ xfwm4
               \_ xfce4-panel
                    \_ xterm
                    |    \_ -tcsh
                    |         \_ acroread
                    \_ xterm
                    |    \_ -tcsh
                    \_ firefox
```

Here, the computer is using lightdm as the display manager, running an X session script named "xinitrc" and using the xfwm4 window manager.

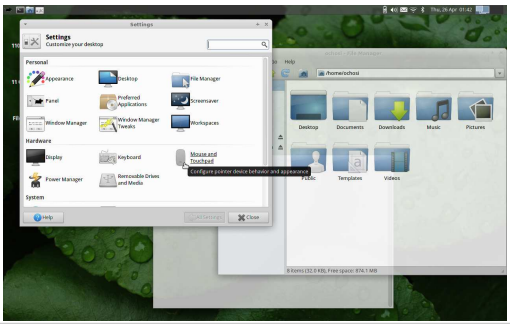
Desktop Environments:



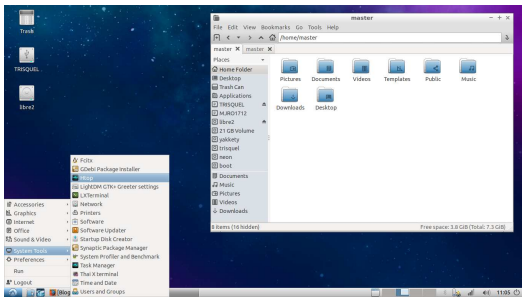
KDE

Gnome

Xfce



Lxde



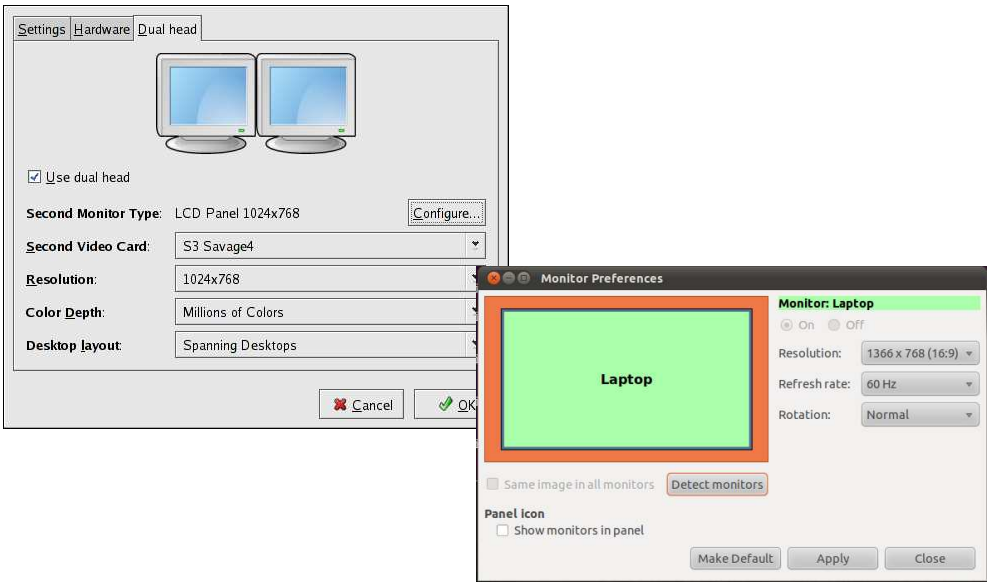
A “desktop environment” is a suite of X applications that have a unified look and feel, and are designed to work well together.

The illustration above shows some of the most popular desktop environments right now, but there are many others.

It's important to remember that there's no reason you can't use applications from different desktop environments at the same time. The X server doesn't care. The choice of desktop environment is primarily primarily about the "look and feel" of the applications.

Configuring the X Server:

Usually, you can configure the X server using graphical tools provided by your Linux distribution. Here are a couple of examples:



Beyond the graphical tools for configuring your display, you might someday need to dig deeper using command-line tools like xdpyinfo, xset, xlsclients, and xrandr. The last of these, in particular, will give you a lot of information about all of the display devices connected to your computer.

X Servers for Windows and OS X:

vcxsrv: https://sourceforge.net/projects/vcxsrv/files/vcxsrv/

Xquartz: https://www.xquartz.org/

X servers are available for MS Windows and Apple OS X. At the moment, I recommend using "vcxsrv" under Windows. This is a free package that isn't officially supported by Microsoft, but it works very well. Apple itself provides the Xquartz X server.

Neither of these entirely controls your display, as an X server would do when used with Linux. Instead, these X servers allow you to display remote X clients on your Windows or OS X desktop.

Part 3: Security

This is the first time we've talked about programs that use the network, so it's time to start thinking about security. Any network-aware software can open your computer up to abuse, if improperly used.

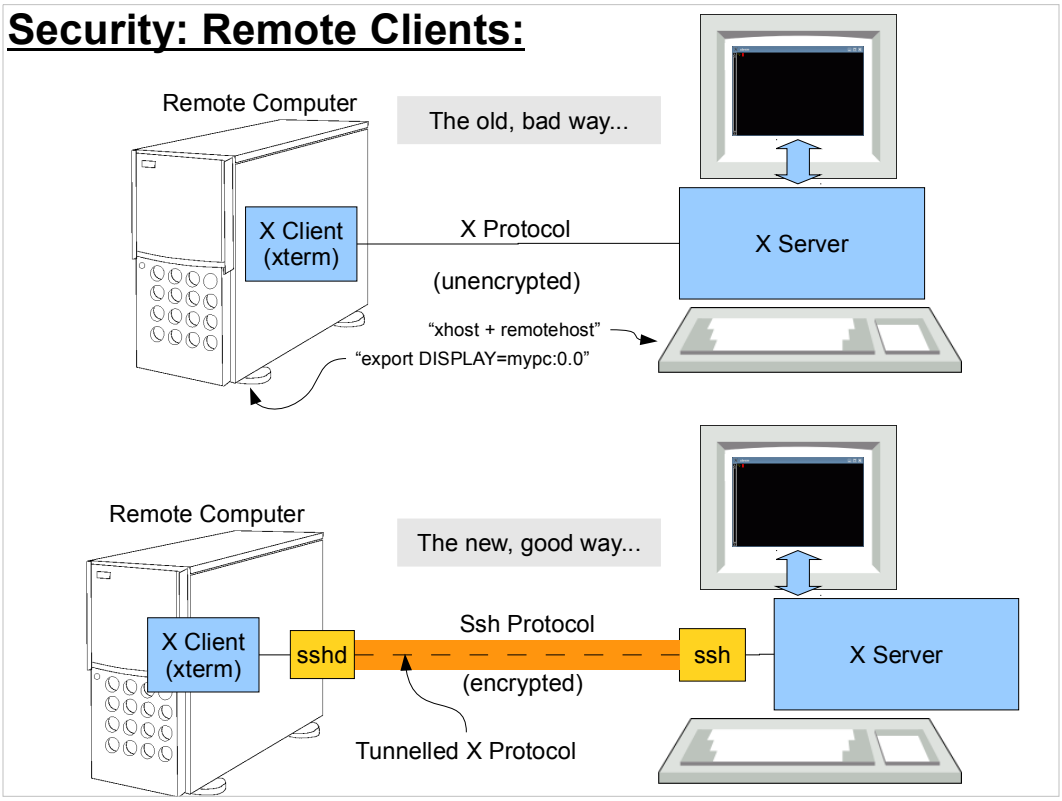
In the following, I'll mention a program called “ssh”, which allows you to get command-line access to another computer on the network. Don't worry about the details of this right now. We'll talk more about it at a later time. The main point is that ssh creates an encrypted connection between the two computers, so that nobody can snoop on your communications.

Sending Commands Securely to Remote Hosts:

ssh is a tool that lets you securely connect to a remote computer, and issue command-line commands there.

When you ssh into a remote computer, you are talking to a command-line shell running on that computer.

We'll talk much more about ssh in a later meeting.



In the bad old days we used to use X in the way shown in the top picture. We'd use an xhost command on the local computer to allow a specific remote computer access to our X server, and then on the remote computer we'd set the DISPLAY environment variable to point to our local computer's X display.

This is insecure for a couple of reasons: (1) the X protocol connection between the computers is unencrypted, and (2) the "xhost" command allows any process on the remote computer (which may not be completely trustworthy) to read our local keystrokes.

The better way to do it is shown in the bottom illustration. Just let ssh set up an encrypted, tunnelled X connection between the machines. This is easy to do, and it Just Works.

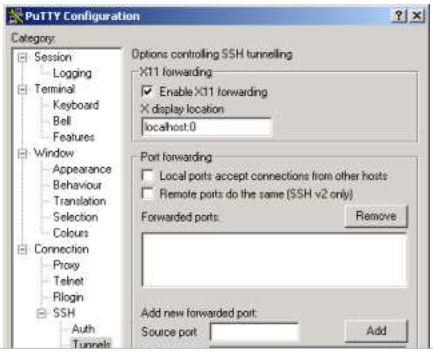
Enabling X Tunnelling for Ssh:

With luck, the default settings for your ssh client will already be configured properly, so that all you need to do is ssh into the remote host, and type "xterm" (or whatever X application you want to run). If this doesn't work, take a look at your ssh client's configuration:

- OpenSSH (Linux and OS X):
  - You can tell ssh to create an X tunnel by typing "ssh -Y" instead of just "ssh".
  - Alternatively, look in the file /etc/ssh/ssh_config or ~/.ssh/config and make sure it contains lines like:

```
ForwardX11 yes
ForwardX11Trusted yes
```

- Putty (MS Windows):



Here's how to set up ssh so that it creates an encrypted, tunnelled X connection between the local and remote computers. Once the defaults are set properly, you shouldn't need to do anything other than ssh to the remote computer and start invoking X applications.

Putty is a free program that you can download from various web sites (just do a Google search for it).

OpenSSH is pre-installed on OS X computers and most Linux computers.

Rules of Thumb for X Security:

1. You should never need to set the DISPLAY variable by hand.

2. You should never need to use xhost.

Beware of documentation that tells you to do these things!

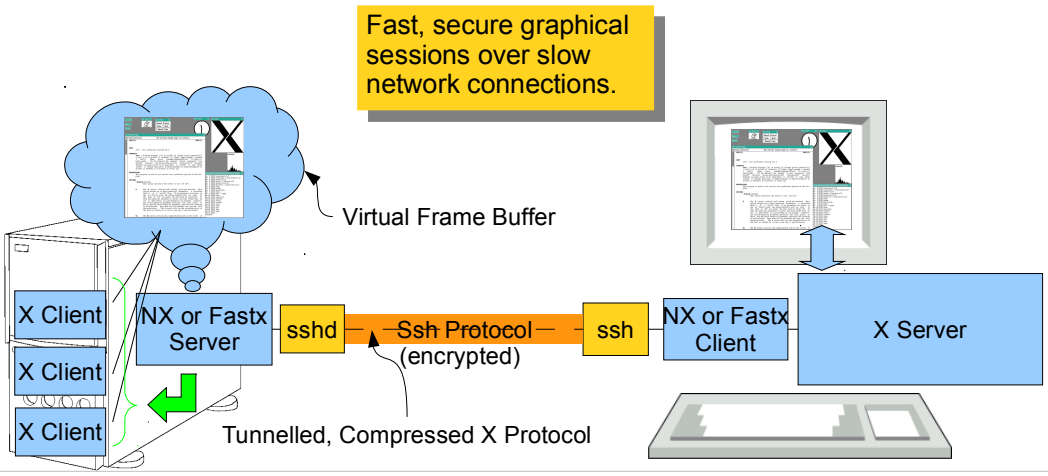
If you don't remember anything else from this talk, remember these two things.

The xhost warning is particularly important. One of the most dangerous commands you can type is “xhost +”, which allows any host in the world to connect to your X server and read your keystrokes. This used to be common practice. (Some X server software for Windows even came pre-configured to do this automatically.) Many years ago, this caused a major security breach at our university. Please don't do that!

Security: Remote Sessions:

https://www.starnet.com/fastx/

http://opennx.net/



So, what if you want to start up a whole session running on a remote computer? The best way to do this is by using a tool like OpenNX or Fastx.

The OpenNX is a free program available for Windows, Linux and OS X. It talks to a remote "NX server" that maintains a local virtual display in memory, called a "Virtual Frame Buffer" (VFB). The NX server starts up applications which send their display information to the VFB. The VFB can then be displayed locally by the nxclient application. All traffic between server and client is tunnelled through an encrypted SSH connection.

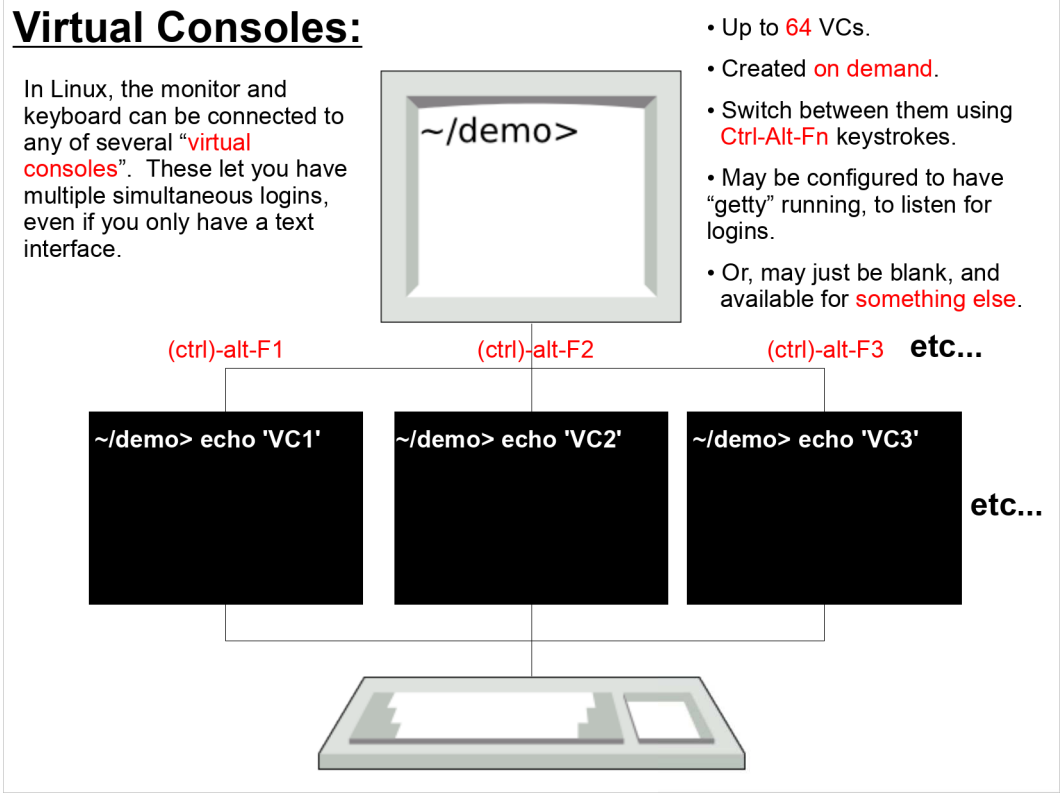
NX has the added benefit that its NX protocol is a highly compressed version of the X protocol, and will give fast performance even over a slow network connection.

FastX is a commercial product that provides similar functionality. At UVa, we currently use FastX to establish graphical connections to our supercomputing cluster, rivanna.

Part 4: Virtual Consoles

One of the simple advantages of a graphical interface is that you can have several terminal windows open at the same time. This can be very convenient if you're working on several different things at once.

Under Linux, you can accomplish something similar without a graphical interface, too. The Linux kernel implements something called “virtual consoles”. These allow one monitor and one keyboard to act like several independent text-based terminals.

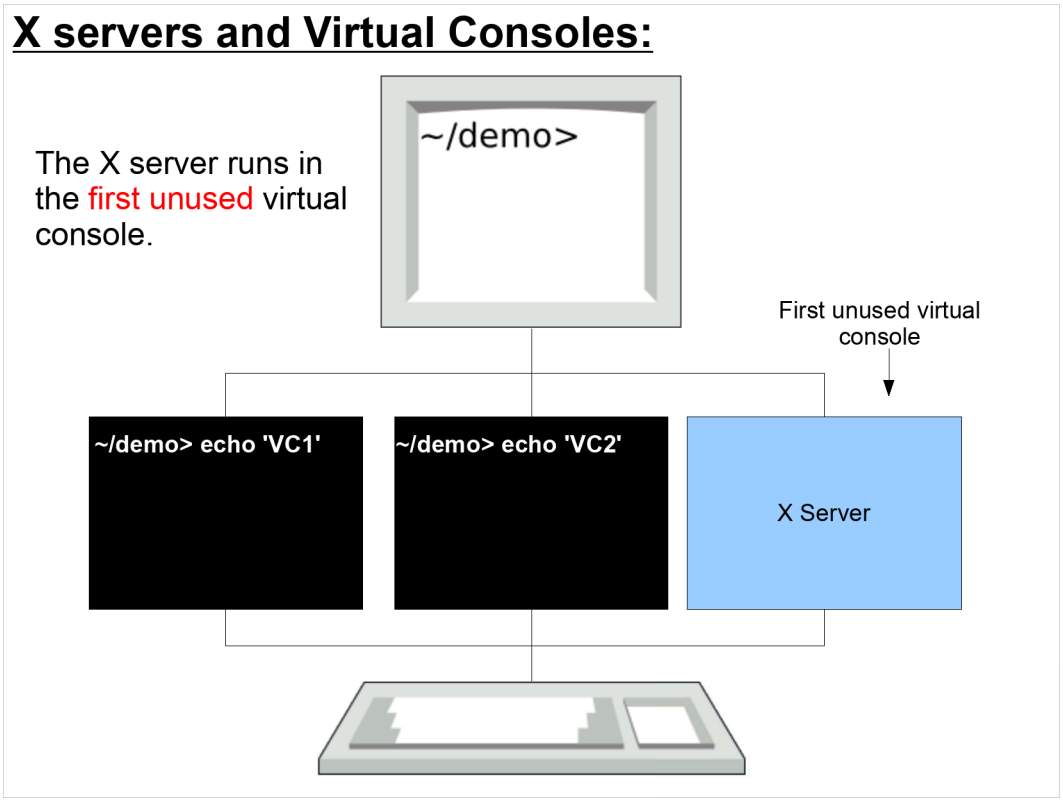


To understand how X works under Linux, it's useful to understand Linux's idea of "virtual consoles". A virtual console is an abstraction, created by the kernel, that can be connected to a real display and keyboard. The kernel creates virtual consoles on demand (up to 64 of them).

Normally, the program "getty" is running on the first few virtual consoles. The "getty" program is the thing that asks you for your username and password, and spawns off a login shell for you when you log in.

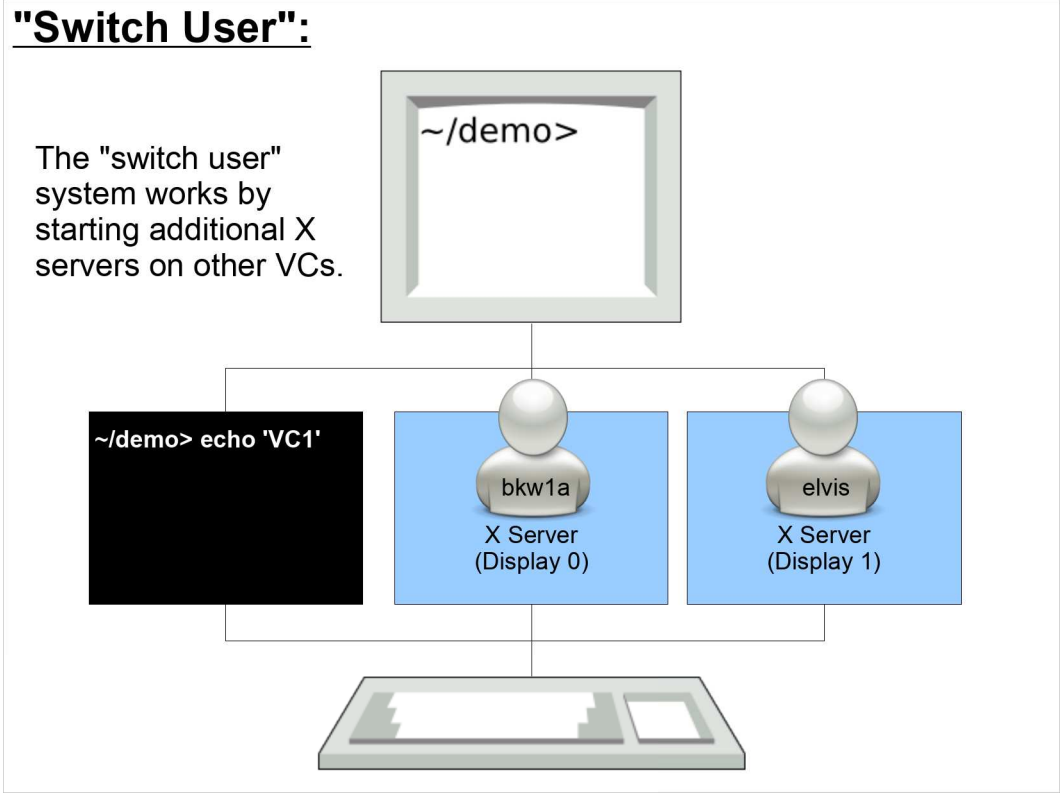
Even if you only have a text interface, you can still have access to multiple simultaneous terminal sessions, through "virtual consoles".

When you're outside of X, you can switch to a different virtual console by typing "alt-Fn", but when you're inside X, you'll need to type "ctrl-alt-Fn", because X typically traps the "alt-Fn" sequence for its own use.



When the “X” program starts, it takes over the first unused virtual console. On most Red Hat-based systems (including Red Hat Enterprise Linux, Fedora, CentOS, etc.), this will be virtual console number 7. On older Ubuntu systems this will be virtual console number 3, or number 7 on newer systems.

While X is running, you can switch to another virtual console by typing “ctrl-alt-Fn”, where n is the virtual console number.



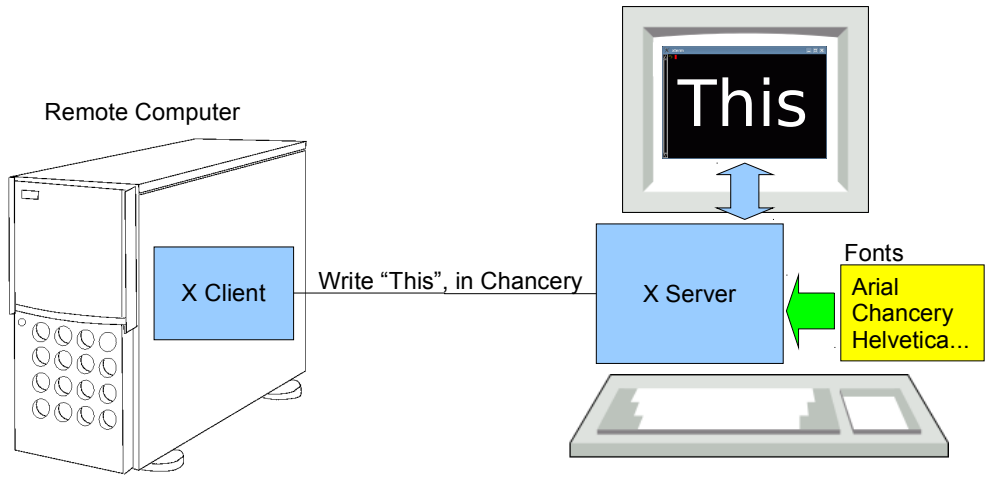
When you "switch users" a new X server is started up on the next unused virtual console, and the display manager presents a login box there. This way, the original user's session can stay running on the first X server.

Part 5: Fonts

This is one of those things you seldom need to think about, but you may someday find that you need to debug a font problem, so let's take a quick look at the two font systems commonly in use under Linux.

Server-Side Fonts (the old way):

Originally, X used server-side fonts. When an application wanted to draw some text, it would send the text to the X server, and tell the server the name of the font it should use to display the text.



The advantage of this is that very little data needs to be passed across the network. This is important for slow networks.

The disadvantage is that the server's fonts might not exactly match what the client expects, and the client can't control exactly what the displayed text really looks like.

Configuring Server-Side Fonts:

Server-side fonts are usually implement through a “font server”, called “xfs”, started at boot time. The font server supplies font information to the X server on demand. It's capable of dealing with several different types of fonts, and translating them into a format that the X server can use. The main xfs configuration file is usually /etc/X11/fs/config .

```
# allow a max of 10 clients to connect to this font server
client-limit = 10

# when a font server reaches its limit, start up a new one
clone-self = on

# where to look for fonts
catalogue = /usr/share/X11/fonts/misc:unscaled,
        /usr/share/X11/fonts/75dpi:unscaled,
        /usr/share/X11/fonts/100dpi:unscaled,
        /usr/share/X11/fonts/Type1,
        /usr/share/X11/fonts/TTF,
        /usr/share/X11/fonts/OTF,
        /usr/share/fonts/default/Type1,
...
# in 12 points, decipoints
default-point-size = 120

# 75 x 75 and 100 x 100
default-resolutions = 75,75,100,100
```

To add a font, it should generally be sufficient to drop the font file into one of the “catalogue” directories and restart xfs.

Listing Server-Side Fonts:

The “xlsfonts” command will give you a list of all the fonts known to the X server:

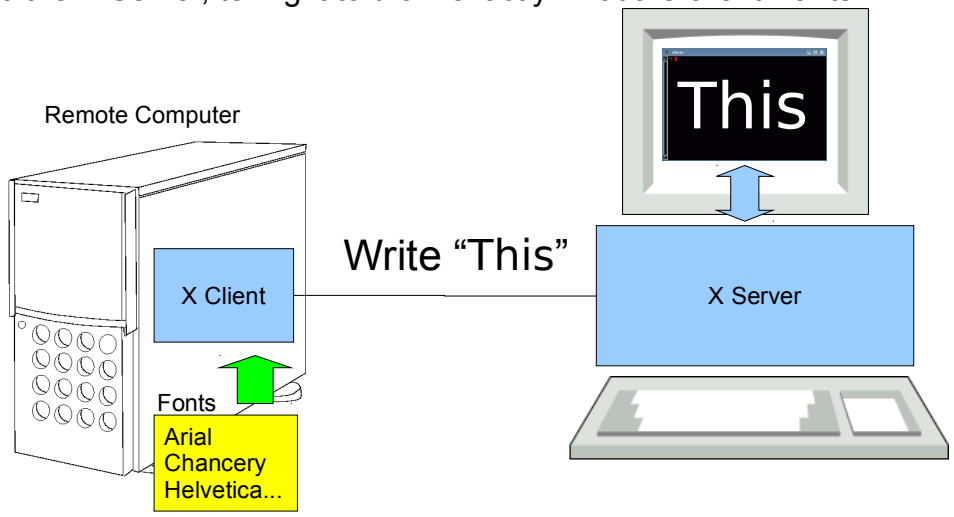
```
~/demo> xlsfonts
-adobe-courier-bold-o-normal--0-0-100-100-m-0-iso8859-2
-adobe-courier-bold-o-normal--0-0-100-100-m-0-iso8859-2
-adobe-courier-bold-o-normal--0-0-75-75-m-0-iso8859-2
-adobe-courier-bold-o-normal--0-0-75-75-m-0-iso8859-2
-adobe-courier-bold-o-normal--0-0-75-75-m-0-koi8-ub
-adobe-courier-bold-o-normal--10-100-75-75-m-60-iso10646-1
-adobe-courier-bold-o-normal--10-100-75-75-m-60-iso8859-1
-adobe-courier-bold-o-normal--10-100-75-75-m-60-iso8859-14
-adobe-courier-bold-o-normal--10-100-75-75-m-60-iso8859-15
-adobe-courier-bold-o-normal--10-100-75-75-m-60-iso8859-2
-adobe-courier-bold-o-normal--10-100-75-75-m-60-iso8859-2
-adobe-courier-bold-o-normal--10-100-75-75-m-60-iso8859-2
-adobe-courier-bold-o-normal--10-100-75-75-m-60-iso8859-9
-adobe-courier-bold-o-normal--10-100-75-75-m-60-koi8-r
-adobe-courier-bold-o-normal--10-100-75-75-m-60-koi8-r
-adobe-courier-bold-o-normal--10-100-75-75-m-60-koi8-u
-adobe-courier-bold-o-normal--10-100-75-75-m-60-koi8-u

...
```

Client-Side Fonts (the new way):

Server-side fonts are now deprecated. Instead, applications should use client-side fonts. Current versions of most major software packages now use client-side fonts.

With client-side fonts, the X client has direct access to the font files. The client lays out the text as it wishes, then sends graphical commands to the X Server, telling it to draw exactly what the client wants.



Current versions of KDE, Gnome, the Mozilla browsers (firefox, seamonkey) and Thunderbird all use client-side fonts. Only a few applications still use server-side fonts.

Configuring Client-Side Fonts:

Client-side fonts are configured through a system called “fontconfig”. The main fontconfig configuration file is /etc/fonts/fonts.conf . This is an XML file containing, among other things, a list of font directories:

```
<!-- Font directory list -->

        <dir>/usr/share/fonts</dir>
        <dir>/usr/share/X11/fonts/Type1</dir>
        <dir>/usr/share/X11/fonts/OTF</dir>
        <dir>~/.fonts</dir>
```

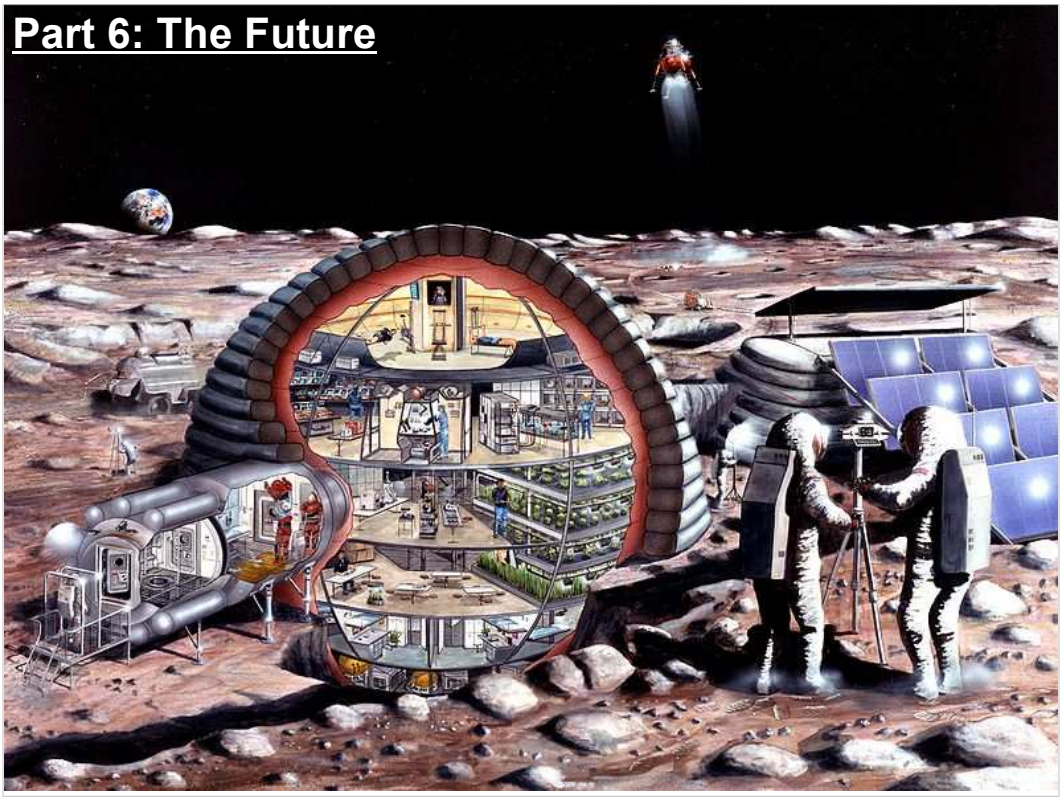
To add a font, the font file should be put into one of these directories (e.g., ~/.fonts) and then the “fc-cache” command should be used to update the fontconfig cache:

```
~/demo> fc-cache -f -v ~/.fonts
```

To get a list of the fonts known to fontconfig, use the “fc-list” command:

```
~/demo> fc-list
Verdana:style=Regular,Normal,Standard
Dustismo:style=Regular
Candara:style=Italic
Luxi Serif:style=Regular
Liberation Mono:style=Regular
MiscFixed:style=Regular
Utopia:style=Bold Italic
```

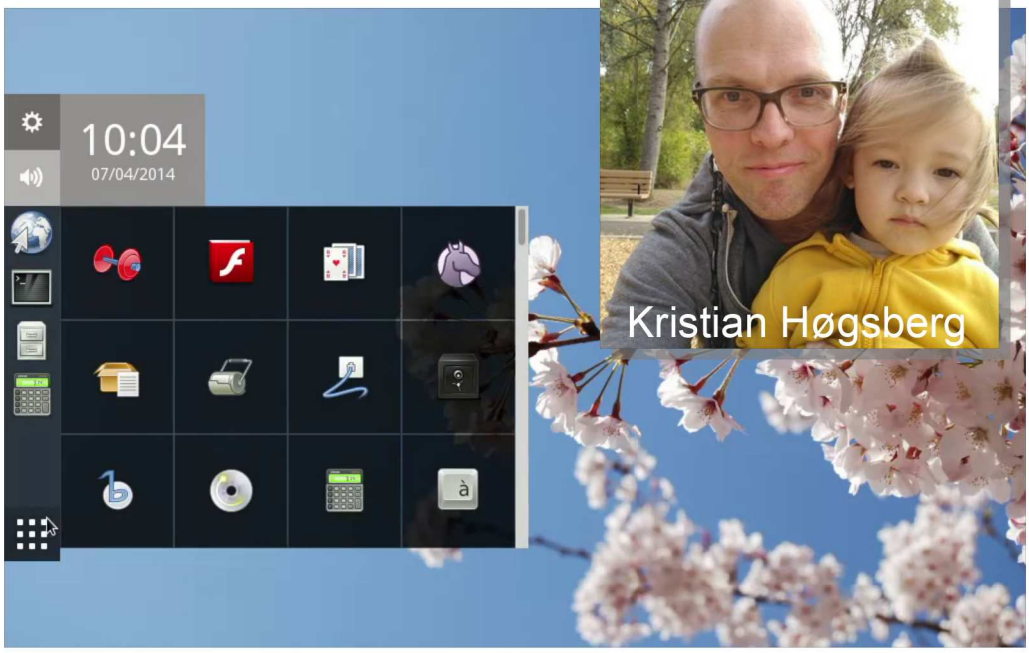
Part 6: The Future



We've talked a little about the shift from "init" to "systemd" for system startup. Another major change is on the horizon, although not yet widely used.

Some Linux distributions have begun to replace the X window system with a successor known as Wayland.

Wayland:



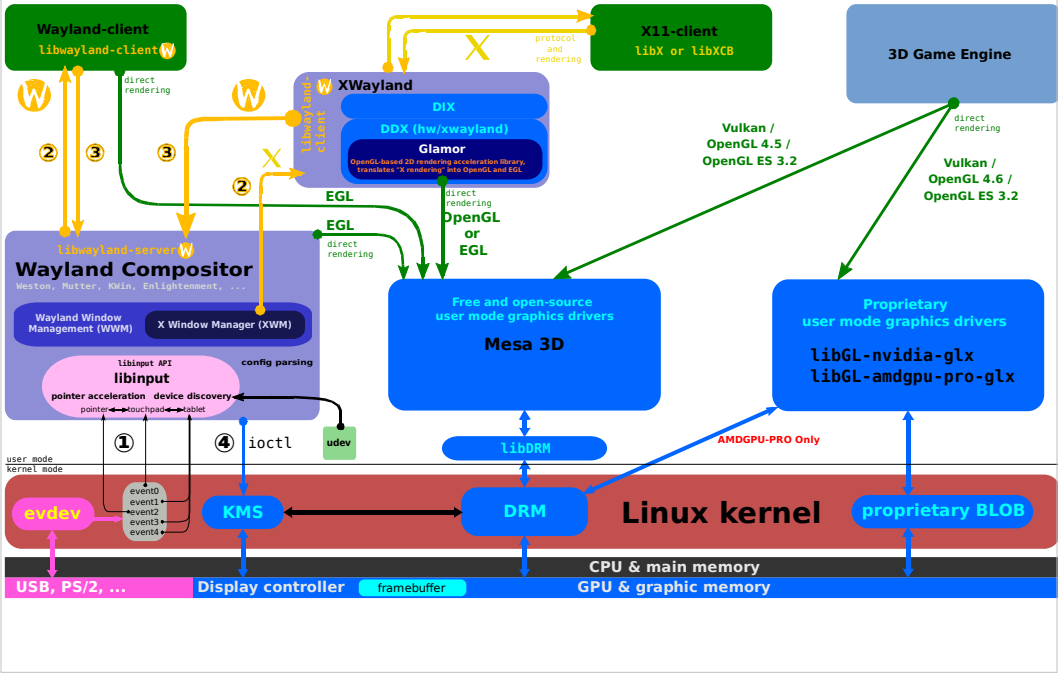
In recent years, Kristian Hogsberg (a programmer who works on graphics software for Linux) has developed an alternative to the X server. He calls his new server “Wayland”. This new graphics system has been written from scratch, without inheriting any of the 20-year-old baggage from X. Hogsberg aims to make Wayland simple, fast and light. It uses existing features in the Linux kernel and modern video chipsets to do most of the work.

Wayland is controversial though, because it doesn't natively support existing X clients and (more controversial yet) it doesn't have network transparency. Nonetheless, one major linux distribution (Fedora) has recently switched to wayland.

For more information about Wayland, see:

http://en.wikipedia.org/wiki/Wayland_(display_server_protocol)

X Graphics with Wayland:



In order for X applications to be used under Wayland, you need an intermediary called "Xwayland". This acts as a traditional X server with respect to the clients, but instead of directly displaying their output, it passed the clients' requests along to Wayland.

With this arrangement, native Wayland and X applications can coexist.

Thanks!