Chapter 5

Advanced Features

This chapter will introduce you to some of the more interesting features of Linux. This assumes that you have at least basic UNIX experience, and understand the information contained in the previous chapters.

The most important aspect of Linux that distinguishes it from other implementations of UNIX is its open design and philosophy. Linux was not developed by a small team of programmers headed by a marketing committee with a single goal in mind. It was developed by an ever-increasing group of hackers, putting what they wanted into a homebrew UNIX system. The types of software and diversity of design in the Linux world is large. Some people dislike this lack of uniformity and conformity—however, some call it one of the strongest qualities of Linux.

5.1 The X Window System

The X Window System is a large and powerful (and somewhat complex) graphics environment for UNIX systems. The original X Window System code was developed at MIT; commercial vendors have since made X the industry standard for UNIX platforms. Virtually every UNIX workstation in the world runs some variant of the X Window system.

A free port of the MIT X Window System version 11, release 6 (X11R6) for 80386/80486/Pentium UNIX systems has been developed by a team of programmers originally headed by David Wexelblat[1]. The release, known as XFree86[2], is available for System V/386, 386BSD, and other x86 UNIX implementations, including Linux. It includes all of the required binaries, support files, libraries, and tools.

Configuring and using the X Window System is far beyond the scope of this book. You are encouraged to read *The X Window System: A User's Guide*—see Appendix A for information on this book. In this section, we'll give a step-by-step description of how to install and configure XFree86 for Linux, but you will have to fill in some of the details yourself by reading the documentation released

[1] David may be reached on the Internet at `dwex@XFree86.org`.

[2] XFree86 is a trademark of The XFree86 Project, Inc.

with XFree86 itself. (This documentation is discussed below.) The Linux *XFree86 HOWTO* is another good source of information.

5.1.1 Hardware requirements

As of XFree86 version 3.1, released in September 1994, the following video chipsets are supported. The documentation included with your video adaptor should specify the chipset used. If you are in the market for a new video card, or are buying a new machine that comes with a video card, have the vendor find out exactly what the make, model, and chipset of the video card is. This may require the vendor to call technical support on your behalf; in general vendors will be happy to do this. Many PC hardware vendors will state that the video card is a "standard SVGA card" which "should work" on your system. Explain that your software (mention Linux and XFree86!) does not support all video chipsets and that you must have detailed information.

You can also determine your videocard chipset by running the `SuperProbe` program included with the XFree86 distribution. This is covered in more detail below.

The following standard SVGA chipsets are supported:

- Tseng ET3000, ET4000AX, ET4000/W32
- Western Digital/Paradise PVGA1
- Western Digital WD90C00, WD90C10, WD90C11, WD90C24, WD90C30, WD90C31, WD90C33
- Genoa GVGA
- Trident TVGA8800CS, TVGA8900B, TVGA8900C, TVGA8900CL, TVGA9000, TVGA9000i, TVGA9100B, TVGA9200CX, TVGA9320, TVGA9400CX, TVGA9420
- ATI 18800, 18800-1, 28800-2, 28800-4, 28800-5, 28800-6, 68800-3, 68800-6, 68800AX, 68800LX, 88800
- NCR 77C22, 77C22E, 77C22E+
- Cirrus Logic CLGD5420, CLGD5422, CLGD5424, CLGD5426, CLGD5428, CLGD5429, CLGD5430, CLGD5434, CLGD6205, CLGD6215, CLGD6225, CLGD6235, CLGD6420
- Compaq AVGA
- OAK OTI067, OTI077
- Avance Logic AL2101
- MX MX68000, MX680010
- Video 7/Headland Technologies HT216-32

The following SVGA chipsets with accelerated features are also supported:

- 8514/A (and true clones)
- ATI Mach8, Mach32
- Cirrus CLGD5420, CLGD5422, CLGD5424, CLGD5426, CLGD5428, CLGD5429, CLGD5430, CLGD5434, CLGD6205, CLGD6215, CLGD6225, CLGD6235
- S3 86C911, 86C924, 86C801, 86C805, 86C805i, 86C928, 86C864, 86C964
- Western Digital WD90C31, WD90C33
- Weitek P9000
- IIT AGX-014, AGX-015, AGX-016
- Tseng ET4000/W32, ET4000/W32i, ET4000/W32p

Video cards using these chipsets are supported on all bus types, including VLB and PCI.

All of the above are supported in both 256 color and monochrome modes, with the exception of the Avance Logic, MX and Video 7 chipsets, which are only supported in 256 color mode. If your video card has enough DRAM installed, many of the above chipsets are supported in 16 and 32 bits-per-pixel mode (specifically, some Mach32, P9000, S3 and Cirrus boards). The usual configuration is 8 bits per pixel (that is, 256 colors).

The monochrome server also supports generic VGA cards, the Hercules monochrome card, the Hyundai HGC1280, Sigma LaserView, and Apollo monochrome cards. On the Compaq AVGA, only 64k of video memory is supported for the monochrome server, and the GVGA has not been tested with more than 64k.

This list will undoubtedly expand as time passes. The release notes for the current version of XFree86 should contain the complete list of supported video chipsets.

One problem faced by the XFree86 developers is that some video card manufacturers use non-standard mechanisms for determining clock frequencies used to drive the card. Some of these manufacturers either don't release specifications describing how to program the card, or they require developers to sign a non-disclosure statement to obtain the information. This would obviously restrict the free distribution of the XFree86 software, something that the XFree86 development team is not willing to do. For a long time, this has been a problem with certain video cards manufactured by Diamond, but as of release 3.1 of XFree86, Diamond has started to work with the development team to release free drivers for these cards.

The suggested setup for XFree86 under Linux is a 486 machine with at least 8 megabytes of RAM, and a video card with a chipset listed above. For optimal performance, we suggest using an accelerated card, such as an S3-chipset card. You should check the documentation for XFree86 and verify that your particular card is supported before taking the plunge and purchasing expensive hardware. Benchmark ratings comparisons for various video cards under XFree86 are posted routinely to the USENET newsgroups `comp.windows.x.i386unix` and `comp.os.linux.misc`.

As a side note, my personal Linux system is a 486DX2-66, 20 megabytes of RAM, and is equipped with a VLB S3-864 chipset card with 2 megabytes of DRAM. I have run X benchmarks on this machine as well as on Sun Sparc IPX workstations. The Linux system is roughly 7 times faster than the Sparc IPX (for the curious, XFree86-3.1 under Linux, with this video card, runs at around 171,000 xstones; the Sparc IPX at around 24,000). In general, XFree86 on a Linux system with an accelerated SVGA card will give you much greater performance than that found on commercial UNIX workstations (which usually employ simple framebuffers for graphics).

Your machine will need at least 4 megabytes of physical RAM, and 16 megabytes of virtual RAM (for example, 8 megs physical and 8 megs swap). Remember that the more physical RAM that you have, the less that the system will swap to and from disk when memory is low. Because swapping is inherently slow (disks are very slow compared to memory), having 8 megabytes of RAM or more is necessary to run XFree86 comfortably. A system with 4 megabytes of physical RAM could run *much* (up to 10 times) more slowly than one with 8 megs or more.

5.1.2 Installing XFree86

The Linux binary distribution of XFree86 can be found on a number of FTP sites. On `sunsite.unc.edu`, it is found in the directory `/pub/Linux/X11`. (As of the time of this writing, the current version is 3.1; newer versions are released periodically).

It's quite likely that you obtained XFree86 as part of a Linux distribution, in which case downloading the software separately is not necessary.

If you are downloading XFree86 directly, This table lists the files in the XFree86-3.1 distribution.

One of the following servers is required:

File	Description
`XF86-3.1-8514.tar.gz`	Server for 8514-based boards.
`XF86-3.1-AGX.tar.gz`	Server for AGX-based boards.
`XF86-3.1-Mach32.tar.gz`	Server for Mach32-based boards.
`XF86-3.1-Mach8.tar.gz`	Server for Mach8-based boards.
`XF86-3.1-Mono.tar.gz`	Server for monochrome video modes.
`XF86-3.1-P9000.tar.gz`	Server for P9000-based boards.
`XF86-3.1-S3.tar.gz`	Server for S3-based boards.
`XF86-3.1-SVGA.tar.gz`	Server for Super VGA-based boards.
`XF86-3.1-VGA16.tar.gz`	Server for VGA/EGA-based boards.
`XF86-3.1-W32.tar.gz`	Server for ET4000/W32-based boards.

All of the following files are required:

File	Description
`XF86-3.1-bin.tar.gz`	The rest of the X11R6 binaries.
`XF86-3.1-cfg.tar.gz`	Config files for `xdm`, `xinit` and `fs`.
`XF86-3.1-doc.tar.gz`	Documentation and manpages.
`XF86-3.1-inc.tar.gz`	Include files.
`XF86-3.1-lib.tar.gz`	Shared X libraries and support files.
`XF86-3.1-fnt.tar.gz`	Basic fonts.

The following files are optional:

File	Description
XF86-3.1-ctrb.tar.gz	Selected contrib programs.
XF86-3.1-extra.tar.gz	Extra XFree86 servers and binaries.
XF86-3.1-lkit.tar.gz	Server linkkit for customization.
XF86-3.1-fnt75.tar.gz	75-dpi screen fonts.
XF86-3.1-fnt100.tar.gz	100-dpi screen fonts.
XF86-3.1-fntbig.tar.gz	Large Kanji and other fonts.
XF86-3.1-fntscl.tar.gz	Scaled fonts (Speedo, Type1).
XF86-3.1-man.tar.gz	Manual pages.
XF86-3.1-pex.tar.gz	PEX binaries, includes and libraries.
XF86-3.1-slib.tar.gz	Static X libraries and support files.
XF86-3.1-usrbin.tar.gz	Daemons which reside in /usr/bin.
XF86-3.1-xdmshdw.tar.gz	Shadow password version of xdm.

The XFree86 directory should contain `README` files and installation notes for the current version.

All that is required to install XFree86 is to obtain the above files, create the directory `/usr/X11R6` (as `root`), and unpack the files from `/usr/X11R6` with a command such as:

```
# gzip -dc XF86-3.1-bin.tar.gz | tar xfB -
```

Remember that these tar files are packed relative to `/usr/X11R6`, so it's important to unpack the files there.

After unpacking the files, you first need to link the file `/usr/X11R6/bin/X` to the server that you're using. For example, if you wish to use the SVGA color server, `/usr/bin/X11/X` should be linked to `/usr/X11R6/bin/XF86_SVGA`. If you wish to use the monochrome server instead, relink this file to `XF86_MONO` with the command

```
# ln -sf /usr/X11R6/bin/XF86_MONO /usr/X11R6/bin/X
```

The same holds true if you are using one of the other servers.

If you aren't sure which server to use, or don't know your video card chipset, you can run the `SuperProbe` program found in `/usr/X11R6/bin` (included in the `XF86-3.1-bin` listed above). This program will attempt to determine your video chipset type and other information; write down its output for later reference.

You need to make sure that `/usr/X11R6/bin` is on your path. This can be done by editing your system default `/etc/profile` or `/etc/csh.login` (based on the shell that you, or other users on your system, use). Or you can simply add the directory to your personal path by modifying `/etc/.bashrc` or `/etc/.cshrc`, based on your shell.

You also need to make sure that `/usr/X11R6/lib` can be located by `ld.so`, the runtime linker. To do this, add the line

```
/usr/X11R6/lib
```

to the file `/etc/ld.so.conf`, and run `/sbin/ldconfig`, as `root`.

5.1.3 Configuring XFree86

Setting up XFree86 is not difficult in most cases. However, if you happen to be using hardware for which drivers ar under development, or wish to obtain the best performance or resolution from an accelerated graphics card, configuring XFree86 can be somewhat time-consuming.

In this section we will describe how to create and edit the `XF86Config` file, which configures the XFree86 server. In many cases it is best to start out with a "basic" XFree86 configuration, one which uses a low resolution, such as 640x480, which should be supported on all video cards and monitor types. Once you have XFree86 working at a lower, standard resolution, you can tweak the configuration to exploit the capabilities of your video hardware. The idea is that you want to know that XFree86 works at all on your system, and that something isn't wrong with your installation, before attempting the sometimes difficult task of setting up XFree86 for real use.

In addition to the information listed here, you should read the following documentation:

- The XFree86 documentation in `/usr/X11R6/lib/X11/doc` (contained within the `XFree86-3.1-doc` package). You should especially see the file `README.Config`, which is an XFree86 configuration tutorial.
- Several video chipsets have separate `README` files in the above directory (such as `README.Cirrus` and `README.S3`). Read one of these if applicable.
- The man page for `XFree86`.
- The man page for `XF86Config`.
- The man page for the particular server that you are using (such as `XF86_SVGA` or `XF86_S3`).

The main XFree86 configuration file is `/usr/X11R6/lib/X11/XF86Config`. This file contains information on your mouse, video card parameters, and so on. The file `XF86Config.eg` is provided with the XFree86 distribution as an example. Copy this file to `XF86Config` and edit it as a starting point.

The `XF86Config` man page explains the format of this file in detail. Read this man page now, if you have not done so already.

We are going to present a sample `XF86Config` file, piece by piece. This file may not look exactly like the sample file included in the XFree86 distribution, but the structure is the same.

◇ Note that the `XF86Config` file format may change with each version of XFree86; this information is only valid for XFree86 version 3.1.

◇ Also, you should not simply copy the configuration file listed here to your own system and attempt to use it. Attempting to use a configuration file which doesn't correspond to your hardware could drive the monitor at a frequency which is too high for it; there have been reports of monitors (especially fixed-frequency monitors) being damaged or destroyed by using an incorrectly configured `XF86Config` file. The bottom line is this: Make absolutely sure that your `XF86Config` file corresponds to your hardware before you attempt to use it.

Each section of the `XF86Config` file is surrounded by the pair of lines `Section` "⟨*section-name*⟩"…`EndSection`. The first part of the `XF86Config` file is `Files`, which looks like this:

```
Section "Files"
    RgbPath     "/usr/X11R6/lib/X11/rgb"
    FontPath    "/usr/X11R6/lib/X11/fonts/misc/"
    FontPath    "/usr/X11R6/lib/X11/fonts/75dpi/"
EndSection
```

The `RgbPath` line sets the path to the X11R6 RGB color database, and each `FontPath` line sets the path to a directory containing X11 fonts. In general you shouldn't have to modify these lines; just be sure that there is a `FontPath` entry for each font type that you have installed (that is, for each directory in `/usr/X11R6/lib/X11/fonts`).

The next section is `ServerFlags`, which specifies several global flags for the server. In general this section is empty.

```
Section "ServerFlags"
# Uncomment this to cause a core dump at the spot where a signal is
# received.  This may leave the console in an unusable state, but may
# provide a better stack trace in the core dump to aid in debugging
#    NoTrapSignals

# Uncomment this to disable the <Crtl><Alt><BS> server abort sequence
#    DontZap
EndSection
```

Here, we have all lines within the section commented out.

The next section is `Keyboard`. This should be fairly intuitive.

```
Section "Keyboard"
    Protocol    "Standard"
    AutoRepeat  500 5
    ServerNumLock
EndSection
```

Other options are available as well—see the `XF86Config` file if you wish to modify the keyboard configuration. The above should work for most systems.

The next section is `Pointer` which specifies parameters for the mouse device.

```
Section "Pointer"

    Protocol    "MouseSystems"
    Device      "/dev/mouse"

# Baudrate and SampleRate are only for some Logitech mice
#    BaudRate   9600
```

```
#     SampleRate 150

# Emulate3Buttons is an option for 2-button Microsoft mice
#     Emulate3Buttons

# ChordMiddle is an option for some 3-button Logitech mice
#     ChordMiddle

EndSection
```

The only options that you should concern yourself with now are `Protocol` and `Device`. `Protocol` specifies the mouse *protocol* that your mouse uses (not the make or brand of mouse). Valid types for `Protocol` (under Linux—there are other options available for other operating systems) are:

- `BusMouse`
- `Logitech`
- `Microsoft`
- `MMSeries`
- `Mouseman`
- `MouseSystems`
- `PS/2`
- `MMHitTab`

`BusMouse` should be used for the Logitech busmouse. Note that older Logitech mice should use `Logitech`, but newer Logitech mice use either `Microsoft` or `Mouseman` protocols. This is a case in which the protocol doesn't necessarily have anything to do with the make of the mouse.

`Device` specifies the device file where the mouse can be accessed. On most Linux systems, this is `/dev/mouse`. `/dev/mouse` is usually a link to the appropriate serial port (such as `/dev/cua0`) for serial mice, or to the appropriate busmouse device for busmice. At any rate, be sure that the device file listed in `Device` exists.

The next section is `Monitor`, which specifies the characteristics of your monitor. As with other sections in the `XF86Config` file, there may be more than one `Monitor` section. This is useful if you have multiple monitors connected to a system, or use the same `XF86Config` file under multiple hardware configurations. In general, though, you will need a single `Monitor` section.

```
Section "Monitor"

    Identifier  "CTX 5468 NI"

    # These values are for a CTX 5468NI only! Don't attempt to use
    # them with your monitor (unless you have this model)
```

```
    Bandwidth    60
    HorizSync    30-38,47-50
    VertRefresh  50-90

    # Modes: Name      dotclock  horiz                vert

    ModeLine "640x480"  25       640 664 760 800      480 491 493 525
    ModeLine "800x600"  36       800 824 896 1024     600 601 603 625
    ModeLine "1024x768" 65       1024 1088 1200 1328  768 783 789 818

  EndSection
```

The `Identifier` line is used to give an arbitrary name to the `Monitor` entry. This can be any string; you will use it to refer to the `Monitor` entry later in the `XF86Config` file.

they are listed below.

`HorizSync` specifies the valid horizontal sync frequencies for your monitor, in kHz. If you have a multisync monitor, this can be a range of values (or several comma-separated ranges), as seen above. If you have a fixed-frequency monitor, this will be a list of discrete values, such as:

```
    HorizSync    31.5, 35.2, 37.9, 35.5, 48.95
```

Your monitor manual should list these values in the technical specifications section. If you do not have this information available, you should either contact the manufacturer or vendor of your monitor to obtain it. There are other sources of information, as well;

`VertRefresh` specifies the valid vertical refresh rates (or vertical synchronization frequencies) for your monitor, in Hz. Like `HorizSync` this can be a range or a list of discrete values; your monitor manual should list them.

`HorizSync` and `VertRefresh` are used only to double-check that the monitor resolutions that you specify are in valid ranges. This is to reduce the chance that you will damage your monitor by attempting to drive it at a frequency for which it was not designed.

The `ModeLine` directive is used to specify a single resolution mode for your monitor. The format of `ModeLine` is

`ModeLine` ⟨*name*⟩ ⟨*clock*⟩ ⟨*horiz-values*⟩ ⟨*vert-values*⟩

⟨*name*⟩ is an arbitrary string, which you will use to refer to the resolution mode later in the file. ⟨*dot-clock*⟩ is the driving clock frequency, or "dot clock" associated with the resolution mode. A dot clock is usually specified in MHz, and is the rate at which the video card must send pixels to the monitor at this resolution. ⟨*horiz-values*⟩ and ⟨*vert-values*⟩ are four numbers each which specify when the electron gun of the monitor should fire, and when the horizontal and vertical sync pulses fire during a sweep.

How can you determine the `ModeLine` values for your monitor? The file `VideoModes.doc`, included with the XFree86 distribution, describes in detail how to determine these values for each resolution mode that your monitor supports. First of all, ⟨*clock*⟩ must correspond to one of the dot clock values that your video card can produce. Later in the `XF86Config` file you will specify these clocks; you can only use video modes which have a ⟨*clock*⟩ value supported by your video card.

There are two files included in the XFree86 distribution which may include `ModeLine` data for your monitor. These files are `modeDB.txt` and `Monitors`, both of which are found in `/usr/X11R6/lib/X11/doc`.

You should start with `ModeLine` values for the VESA standard monitor timings, which most monitors support. `modeDB.txt` includes timing values for VESA standard resolutions. In that file, you will see entries such as

```
# 640x480@60Hz Non-Interlaced mode
# Horizontal Sync = 31.5kHz
# Timing: H=(0.95us, 3.81us, 1.59us), V=(0.35ms, 0.064ms, 1.02ms)
#
# name        clock   horizontal timing     vertical timing      flags
 "640x480"     25.175  640  664  760  800    480  491  493  525
```

This is a VESA standard timing for a 640x480 video mode. It uses a dot clock of 25.175, which your video card must support to use this mode (more on this later). To include this entry in the `XF86Config` file, you'd use the line

```
ModeLine "640x480" 25.175  640 664 760 800  480 491 493 525
```

Note that the ⟨*name*⟩ argument to `ModeLine` (in this case `"640x480"`) is an arbitrary string—the convention is to name the mode after the resolution, but ⟨*name*⟩ can technically be anything descriptive which describes the mode to you.

For each `ModeLine` used the server will check that the specifications for the mode fall within the range of values specified with `Bandwidth`, `HorizSync` and `VertRefresh`. If they do not, the server will complain when you attempt to start up X (more on this later). For one thing, the dot clock used by the mode should not be greater than the value used for `Bandwidth`. (However, in many cases it is safe to use modes with a slightly higher bandwidth than your monitor can support.)

If the VESA standard timings do not work for you (you'll know after trying to use them later) then the files `modeDB.txt` and `Monitors` include specific mode values for many monitor types. You can create `ModeLine` entries from the values found in those two files as well. Be sure to only use values for the specific model of monitor that you have. Note that many 14 and 15-inch monitors cannot support higher resolution modes, and often resolutions of 1024x768 at low dot clocks. This means that if you can't find high resolution modes for your monitor in these files, then your monitor probably does not support those resolution modes.

If you are completely at a loss, and can't find working `ModeLine` values for your monitor, you can follow the instructions in the `VideoModes.doc` file included in the XFree86 distribution to generate `ModeLine` values from the specifications listed in your monitor's manual. While your mileage will certainly vary when attempting to generate `ModeLine` values by hand, this is a good place to look if you can't find the values that you need. `VideoModes.doc` also describes the format of the `ModeLine` directive and other aspects of the XFree86 server in gory detail.

Lastly, if you do obtain `ModeLine` values which are almost, but not quite, right, then it may be possible to simply modify the values slightly to obtain the desired result. For example, if while running XFree86 the image on the monitor is shifted slightly, or seems to "roll", you can follow the instructions in the `VideoModes.doc` file to try to fix these values. Also, be sure to check the knobs and controls on the monitor itself! In many cases it is necessary to change the horizontal or vertical size of the display after starting up XFree86 in order for the image to be centered and be of the appropriate size. Having these controls on the front of the monitor can certainly make life easier.

◇ You shouldn't use monitor timing values or `ModeLine` values for monitors other than the model that you own. If you attempt to drive the monitor at a frequency for which it was not designed, you can damage or even destroy it.

The next section of the `XF86Config` file is `Device`, which specifies parameters for your video card. Here is an example.

```
Section "Device"
        Identifier "#9 GXE 64"

        # Nothing yet; we fill in these values later.

EndSection
```

This section defines properties for a particular video card. `Identifier` is an arbitrary string describing the card; you will use this string to refer to the card later.

Initially, you don't need to include anything in the `Device` section, except for `Identifier`. This is because we will be using the X server itself to probe for the properties of the video card, and entering them into the `Device` section later. The XFree86 server is capable of probing for the video chipset, clocks, RAMDAC, and amount of video RAM on the board.

Before we do this, however, we need to finish writing the `XF86Config` file. The next section is `Screen`, which specifies the monitor/video card combination to use for a particular server.

```
Section "Screen"
    Driver     "Accel"
    Device     "#9 GXE 64"
    Monitor    "CTX 5468 NI"
    Subsection "Display"
        Depth      16
        Modes      "1024x768" "800x600" "640x480"
        ViewPort   0 0
        Virtual    1024 768
    EndSubsection
EndSection
```

The `Driver` line specifies the X server that you will be using. The value values for `Driver` are:

- `Accel`: For the `XF86_S3`, `XF86_Mach32`, `XF86_Mach8`, `XF86_8514`, `XF86_P9000`, `XF86_AGX`, and `XF86_W32` servers;

- `SVGA`: For the `XF86_SVGA` server;
- `VGA16`: For the `XF86_VGA16` server;
- `VGA2`: For the `XF86_Mono` server;
- `Mono`: For the non-VGA monochrome drivers in the `XF86_Mono` and `XF86_VGA16` servers.

You should be sure that `/usr/X11R6/bin/X` is a symbolic link to the server that you are using.

The `Device` line specifies the `Identifier` of the `Device` section corresponding to the video card to use for this server. Above, we created a `Device` section with the line

```
Identifier "#9 GXE 64"
```

Therefore, we use `"#9 GXE 64"` on the `Device` line here.

Similarly, the `Monitor` line specifies the name of the `Monitor` section to be used with this server. Here, `"CTX 5468 NI"` is the `Identifier` used in the `Monitor` section described above.

`Subsection "Display"` defines several properties of the XFree86 server corresponding to your monitor/video card combination. The `XF86Config` file describes all of these options in detail; most of them are icing on the cake and not necessary to get the system working.

The options that you should know about are:

- `Depth`. Defines the number of color planes—the number of bits per pixel. Usually, `Depth` is set to 8. For the `VGA16` server, you would use a depth of 4, and for the monochrome server a depth of 1. If you are using an accelerated video card with enough memory to support more bits per pixel, you can set `Depth` to 16, 24, or 32. If you have problems with depths higher than 8, set it back to 8 and attempt to debug the problem later.
- `Modes`. This is the list of video mode names which have been defined using the `ModeLine` directive in the `Monitor` section. In the above section, we used `ModeLines` named `"1024x768"`, `"800x600"`, and `"640x48"0`. Therefore, we use a `Modes` line of

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

  The first mode listed on this line will be the default when XFree86 starts up. After XFree86 is running, you can switch between the modes listed here using the keys `ctrl`-`alt`-`numeric +` and `ctrl`-`alt`-`numeric -`.

  It might be best, when initially configuring XFree86, to use lower resolution video modes, such as 640x480, which tend to work on most systems. Once you have the basic configuration working you can modify `XF86Config` to support higher resolutions.
- `Virtual`. Sets the virtual desktop size. XFree86 has the ability to use any additional memory on your video card to extend the size of your desktop. When you move the mouse pointer to the edge of the display, the desktop will scroll, bringing the additional space into view. Therefore, even if you are running at a lower video resolution such as 800x600, you can set `Virtual` to the total resolution which your video card can support (a 1-megabyte video card

can support 1024x768 at a depth of 8 bits per pixel; a 2-megabyte card 1280x1024 at depth 8, or 1024x768 at depth 16). Of course, the entire area will not be visible at once, but it can still be used.

The `Virtual` feature is a nice way to utilize the memory of your video card, but it is rather limited. If you want to use a true virtual desktop, we suggest using `fvwm`, or a similar window manager, instead. `fvwm` allows you to have rather large virtual desktops (implemented by hiding windows, and so forth, instead of actually storing the entire desktop in video memory at once). See the man pages for `fvwm` for more details about this; most Linux systems use `fvwm` by default.

- `ViewPort`. If you are using the `Virtual` option described above, `ViewPort` sets the coordinates of the upper-left-hand corner of the virtual desktop when XFree86 starts up. `Virtual 0 0` is often used; if this is unspecified then the desktop is centered on the virtual desktop display (which may be undesirable to you).

Many other options for this section exist; see the `XF86Config` man page for a complete description. In practice these other options are not necessary to get XFree86 initially working.

5.1.4 Filling in video card information

Your `XF86Config` file is now ready to go, with the exception of complete information on the video card. What we're going to do is use the X server to probe for the rest of this information, and fill it into `XF86Config`.

Instead of probing for this information with the X server, the `XF86Config` values for many cards are listed in the files `modeDB.txt`, `AccelCards`, and `Devices`. These files are all found in `/usr/X11R6/lib/X11/doc`. In addition, there are various `README` files for certain chipsets. You should look in these files for information on your video card, and use that information (the clock values, chipset type, and any options) in the `XF86Config` file. If any information is missing, you can probe for it as described here.

In these examples we will demonstrate configuration for a #9 GXE 64 video card, which uses the `XF86_S3` chipset. This card happens to be the one which the author uses, but the discussion here applies to any video card.

The first thing to do is to determine the video chipset used on the card. Running `SuperProbe` (found in `/usr/X11R6/bin`) will tell you this information, but you need to know the chipset name as it is known to the X server.

To do this, run the command

```
X -showconfig
```

This will give the chipset names known to your X server. (The man pages for each X server list these as well.) For example, with the accelerated `XF86_S3` server, we obtain:

```
XFree86 Version 3.1 / X Window System
```

```
(protocol Version 11, revision 0, vendor release 6000)
Operating System: Linux
Configured drivers:
  S3: accelerated server for S3 graphics adaptors (Patchlevel 0)
      mmio_928, s3_generic
```

The valid chipset names for this server are `mmio_928` and `s3_generic`. The `XF86_S3` man page describes these chipsets and which videocards use them. In the case of the #9 GXE 64 video card, `mmio_928` is appropriate.

If you don't know which chipset to use, the X server can probe it for you. To do this, run the command

```
X -probeonly > /tmp/x.out 2>&1
```

if you use `bash` as your shell. If you use `csh`, try:

```
X -probeonly &> /tmp/x.out
```

You should run this command while the system is unloaded, that is, while no other activity is occurring on the system. This command will also probe for your video card dot clocks (as seen below), and system load can throw off this calculation.

The output from the above (in `/tmp/x.out` should contain lines such as the following:

```
XFree86 Version 3.1 / X Window System
(protocol Version 11, revision 0, vendor release 6000)
Operating System: Linux
Configured drivers:
  S3: accelerated server for S3 graphics adaptors (Patchlevel 0)
      mmio_928, s3_generic
Several lines deleted...
(--) S3: card type: 386/486 localbus
(--) S3: chipset:   864 rev. 0
(--) S3: chipset driver: mmio_928
```

Here, we see that the two valid chipsets for this server (in this case, `XF86_S3`) are `mmio_928` and `s3_generic`. The server probed for and found a video card using the `mmio_928` chipset.

In the `Device` section of the `XF86Config` file, add a `Chipset` line, containing the name of the chipset as determined above. For example,

```
Section "Device"
        # We already had Identifier here...
        Identifier "#9 GXE 64"
        # Add this line:
        Chipset "mmio_928"
EndSection
```

Now we need to determine the driving clock frequencies used by the video card. A driving clock frequency, or dot clock, is simply a rate at which the video card can send pixels to the monitor. As we have seen, each monitor resolution has a dot clock associated with it. Now we need to determine which dot clocks are made available by the video card.

First you should look into the files (`modeDB.txt`, and so forth) mentioned above and see if your card's clocks are listed there. The dot clocks will usually be a list of 8 or 16 values, all of which are in MHz. For example, when looking at `modeDB.txt` we see an entry for the Cardinal ET4000 video board, which looks like this:

```
# chip    ram   virtual    clocks                                 default-mode  flags
 ET4000   1024  1024 768    25   28   38   36   40   45   32    0  "1024x768"
```

As we can see, the dot clocks for this card are 25, 28, 38, 36, 40, 45, 32, and 0 MHz.

In the `Devices` section of the `XF86Config` file, you should add a `Clocks` line containing the list of dot clocks for your card. For example, for the clocks above, we would add the line

```
Clocks 25 28 38 36 40 45 32 0
```

to the `Devices` section of the file, after `Chipset`. Note that the order of the clocks is important! Don't resort the list of clocks or remove duplicates.

If you cannot find the dot clocks associated with your card, the X server can probe for these as well. Using the `X -probeonly` command described above, the output should contain lines which look like the following:

```
(--) S3: clocks:  25.18  28.32  38.02  36.15  40.33  45.32  32.00  00.00
```

We could then add a `Clocks` line containing all of these values, as printed. You can use more than one `Clocks` line in `XF86Config` should all of the values (sometimes there are more than 8 clock values printed) not fit onto one line. Again, be sure to keep the list of clocks in order as they are printed.

Be sure that there is no `Clocks` line (or that it is commented out) in the `Devices` section of the file when using `X -probeonly` to probe for the clocks. If there is a `Clocks` line present, the server will *not* probe for the clocks—it will use the values given in `XF86Config`.

Note that some accelerated video boards use a programmable clock chip. (See the `XF86_Accel` man page for details; this generally applies to S3, AGX, and XGA-2 boards.) This chip essentially allows the X server to tell the card which dot clocks to use. If this is the case, then you may not find a list of dot clocks for the card in any of the above files. Or, the list of dot clocks printed when using `X -probeonly` will only contain one or two discrete clock values, with the rest being duplicates or zero.

For boards which use a programmable clock chip, you would use a `ClockChip` line, instead of a `Clocks` line, in your `XF86Config` file. `ClockChip` gives the name of the clock chip as used by the video card; the man pages for each server describe what these are. For example, in the file `README.S3`, we see that several S3-864 video cards use an "ICD2061A" clock chip, and that we should use the line

```
ClockChip "icd2061a"
```

instead of Clocks in the XF86Config file. As with Clocks, this line should go in the Devices section, after Chipset.

Similarly, some accelerated cards require you to specify the RAMDAC chip type in the XF86Config file, using a Ramdac line. The XF86_Accel man page describes this option. Usually, the X server will correctly probe for the RAMDAC.

Some video card types require you to specify several options in the Devices section of XF86Config. These options will be described in the man page for your server, as well as in the various files (such as README.cirrus or README.S3. These options are enabled using the Option line. For example, the #9 GXE 64 card requires two options:

```
Option "number_nine"
Option "dac_8_bit"
```

Usually, the X server will work without these options, but they are necessary to obtain the best performance. There are too many such options to list here, and they each depend on the particular video card being used. If you must use one of these options, fear not—the X server man pages and various files in /usr/X11R6/lib/X11/doc will tell you what they are.

So, when you're finished, you should end up with a Devices section which looks something like this:

```
Section "Device"
        # Device section for the #9 GXE 64 only !
        Identifier "#9 GXE 64"
        Chipset "mmio_928"
        ClockChip "icd2061a"
        Option "number_nine"
        Option "dac_8_bit"
EndSection
```

Most video cards will require a Clocks line, instead of ClockChip, as described above. The above Device entry is only valid for a particular video card, the #9 GXE 64. It is given here only as an example.

There are other options that you can include in the Devices entry. Check the X server man pages for the gritty details, but the above should suffice for most systems.

5.1.5 Running XFree86

With your XF86Config file configured, you're ready to fire up the X server and give it a spin. First, be sure that /usr/X11R6/bin is on your path.

The command to start up XFree86 is

```
startx
```

This is a front-end to `xinit` (in case you're used to using `xinit` on other UNIX systems).

This command will start the X server and run the commands found in the file `.xinitrc` in your home directory. `.xinitrc` is just a shell script containing X clients to run. If this file does not exist, the system default `/usr/X11R6/lib/X11/xinit/xinitrc` will be used.

A standard `.xinitrc` file looks like this:

```
#!/bin/sh

xterm -fn 7x13bold -geometry 80x32+10+50 &
xterm -fn 9x15bold -geometry 80x34+30-10 &
oclock -geometry 70x70-7+7 &
xsetroot -solid midnightblue &

exec twm
```

This script will start up two `xterm` clients, an `oclock`, and set the root window (background) color to `midnightblue`. It will then start up `twm`, the window manager. Note that `twm` is executed with the shell's `exec` statement; this causes the `xinit` process to be replaced with `twm`. Once the `twm` process exits, the X server will shut down. You can cause `twm` to exit by using the root menus: depress mouse button 1 on the desktop background—this will display a pop up menu which will allow you to `Exit Twm`.

Be sure that the last command in `.xinitrc` is started with `exec`, and that it is not placed into the background (no ampersand on the end of the line). Otherwise the X server will shut down as soon as it has started the clients in the `.xinitrc` file.

Alternately, you can exit X by pressing `ctrl`-`alt`-`backspace` in combination. This will kill the X server directly, exiting the window system.

The above is a very, very simple desktop configuration. Many wonderful programs and configurations are available with a bit of work on your `.xinitrc` file. For example, the `fvwm` window manager will provide a virtual desktop, and you can customize colors, fonts, window sizes and positions, and so forth to your heart's content. Although the X Window System might appear to be simplistic at first, it is extremely powerful once you customize it for yourself.

If you are new to the X Window System environment, we strongly suggest picking up a book such as *The X Window System: A User's Guide*. Using and configuring X is far too in-depth to cover here. See the man pages for `xterm`, `oclock`, and `twm` for clues on getting started.

5.1.6 Running into trouble

Often, something will not be quite right when you initially fire up the X server. This is almost always caused by a problem in your `XF86Config` file. Usually, the monitor timing values are off, or the video card dot clocks set incorrectly. If your display seems to roll, or the edges are fuzzy, this is a clear indication that the monitor timing values or dot clocks are wrong. Also be sure that you are correctly specifying your video card chipset, as well as other options for the `Device` section of `XF86Config`. Be absolutely certain that you are using the right X server and that `/usr/X11R6/bin/X` is a symbolic link to this server.

If all else fails, try to start X "bare"; that is, use a command such as:

```
X > /tmp/x.out 2>&1
```

You can then kill the X server (using the `ctrl`-`alt`-`backspace` key combination) and examine the contents of `/tmp/x.out`. The X server will report any warnings or errors—for example, if your video card doesn't have a dot clock corresponding to a mode supported by your monitor.

The file `VideoModes.doc` included in the XFree86 distribution contains many hints for tweaking the values in your `XF86Config` file.

Remember that you can use `ctrl`-`alt`-`numeric +` and `ctrl`-`alt`-`numeric -` to switch between the video modes listed on the `Modes` line of the `Screen` section of `XF86Config`. If the highest resolution mode doesn't look right, try switching to lower resolutions. This will let you know, at least, that those parts of your X configuration are working correctly.

Also, check the vertical and horizontal size/hold knobs on your monitor. In many cases it is necessary to adjust these when starting up X. For example, if the display seems to be shifted slightly to one side, you can usually correct this using the monitor controls.

The USENET newsgroup `comp.windows.x.i386unix` is devoted to discussions about XFree86. It might be a good idea to watch that newsgroup for postings relating to your video configuration—you might run across someone with the same problems as your own.

5.2 Accessing MS-DOS Files

If, for some twisted and bizarre reason, you would have need to access files from MS-DOS, it's quite easily done under Linux.

The usual way to access MS-DOS files is to mount an MS-DOS partition or floppy under Linux, allowing you to access the files directly through the filesystem. For example, if you have an MS-DOS floppy in `/dev/fd0`, the command

```
# mount -t msdos /dev/fd0 /mnt
```

will mount it under `/mnt`. See Section 4.6.2 for more information on mounting floppies.

You can also mount an MS-DOS partition of your hard drive for access under Linux. If you have an MS-DOS partition on `/dev/hda1`, the command

```
# mount -t msdos /dev/hda1 /mnt
```

will mount it. Be sure to `umount` the partition when you're done using it. You can have your MS-DOS partitions automatically mounted at boot time if you include entries for them in `/etc/fstab`;

see Section 4.8 for details. For example, the following line in `/etc/fstab` will mount an MS-DOS partition on `/dev/hda1` on the directory `/dos`.

```
/dev/hda1      /dos      msdos       defaults
```

The `Mtools` software may also be used to access MS-DOS files. For example, the commands `mcd`, `mdir`, and `mcopy` all behave as their MS-DOS counterparts. If you installed `Mtools`, there should be man pages available for these commands.

Accessing MS-DOS files is one thing; running MS-DOS programs from Linux is another. There is an MS-DOS Emulator under development for Linux; it is widely available, and even distributed with SLS. It can be retrieved from a number of locations, including the various Linux FTP sites (see Appendix C for details). The MS-DOS Emulator is reportedly powerful enough to run a number of applications, including Wordperfect, from Linux. However, Linux and MS-DOS are vastly different operating systems. The power of any MS-DOS emulator under UNIX is somewhat limited.

In addition, work is underway on a Microsoft Windows emulator to run under X Windows. Watch the newsgroups and FTP sites for more information.

5.3 Networking with TCP/IP

Linux supports a full implementation of the TCP/IP (Transport Control Protocol/Internet Protocol) networking protocols. TCP/IP has become the most successful mechanism for networking computers worldwide. With Linux and an Ethernet card, you can network your machine to a local area network, or (with the proper network connections), to the Internet—the worldwide TCP/IP network.

Hooking up a small LAN of UNIX machines is easy. It simply requires an Ethernet controller in each machine and the appropriate Ethernet cables and other hardware. Or, if your business or university provides access to the Internet, you can easily add your Linux machine to this network.

The current implementation of TCP/IP and related protocols for Linux is called "NET-2". This has no relationship to the so-called NET-2 release of BSD UNIX; instead, "NET-2" in this context means the second implementation of TCP/IP for Linux.

Linux NET-2 also supports SLIP—Serial Line Internet Protocol. SLIP allows you to have dialup Internet access using a modem. If your business or university provides SLIP access, you can dial in to the SLIP server and put your machine on the Internet over the phone line. Alternately, if your Linux machine also has Ethernet access to the Internet, you can set up your Linux box as a SLIP server.

For complete information on setting up TCP/IP under Linux, we encourage you to read the Linux NET-2 HOWTO, available via anonymous FTP from `sunsite.unc.edu`. The NET-2 HOWTO is a complete guide to configuring TCP/IP, including Ethernet and SLIP connections, under Linux. The Linux Ethernet HOWTO is a related document that describes configuration of various Ethernet card drivers for Linux. The *Linux Network Administrator's Guide*, from the Linux Documentation Project, is also available. See Appendix A for more information on these documents.

Also of interest is the book *TCP/IP Network Administration*, by Craig Hunt. It contains complete information on using and configuring TCP/IP on UNIX systems.

5.3.1 Hardware Requirements

You can use Linux TCP/IP without any networking hardware at all—configuring "loopback" mode allows you to talk to yourself. This is necessary for some applications and games which use the "loopback" network device.

However, if you want to use Linux with an Ethernet TCP/IP network, you need one of the following Ethernet cards: 3com 3c503, 3c503/16; Novell NE1000, NE2000; Western Digital WD8003, WD8013; Hewlett Packard HP27245, HP27247, HP27250.

The following clones are reported to work: WD-80x3 clones: LANNET LEC-45; NE2000 clones: Alta Combo, Artisoft LANtastic AE-2, Asante Etherpak 2001/2003, D-Link Ethernet II, LTC E-NET/16 P/N 8300-200-002, Network Solutions HE-203, SVEC 4 Dimension Ethernet, 4-Dimension FD0490 EtherBoard 16, and D-Link DE-600, SMC Elite 16.

See the Linux Ethernet HOWTO for a more complete discussion of Linux Ethernet hardware compatibility.

Linux also supports SLIP, which allows you to use a modem to access the Internet over the phone line. In this case, you'll need a modem compatible with your SLIP server—most servers require a 14.4bps V.32bis modem.

5.3.2 Configuring TCP/IP on your system

In this section we're going to discuss how to configure an Ethernet TCP/IP connection on your system. Note that this method should work for many systems, but certainly not all. This discussion should be enough to get you on the right path to configuring the network parameters of your machine, but there are numerous caveats and fine details not mentioned here. We direct you to the *Linux Network Administrators' Guide* and the NET-2-HOWTO for more information.[3]

First of all, we assume that you have a Linux system that has the TCP/IP software installed. This includes basic clients such as `telnet` and `ftp`, system administration commands such as `ifconfig` and `route` (usually found in `/etc`), and networking configuration files (such as `/etc/hosts`). The other Linux-related networking documents described above explain how to go about installing the Linux networking software if you do not have it already.

We also assume that your kernel has been configured and compiled with TCP/IP support enabled. See Section 4.7 for information on compiling your kernel. To enable networking, you must answer "yes" to the appropriate questions during the `make config` step, and rebuild the kernel.

Once this has been done, you must modify a number of configuration files used by NET-2. For the most part this is a simple procedure. Unfortunately, however, there is wide disagreement between Linux distributions as to where the various TCP/IP configuration files and support programs should

[3]Some of this information is adapted from the NET-2-HOWTO by Terry Dawson and Matt Welsh.

go. Much of the time, they can be found in `/etc`, but in other cases may be found in `/usr/etc`, `/usr/etc/inet`, or other bizarre locations. In the worst case you'll have to use the **find** command to locate the files on your system. Also note that not all distributions keep the NET-2 configuration files and software in the same location—they may be spread across several directories.

The following information applies primarily to Ethernet connections. If you're planning to use SLIP, read this section to understand the concepts, and follow the SLIP-specific instructions in the following section.

5.3.2.1 Your network configuration

Before you can configure TCP/IP, you need to determine the following information about your network setup. In most cases, your local network administrator can provide you with this information.

- IP address. This is the unique machine address in dotted-decimal format. An example is 128.253.153.54. Your network admins will provide you with this number.

  If you're only configuring loopback mode (i.e. no SLIP, no ethernet card, just TCP/IP connections to your own machine) then your IP address is 127.0.0.1.

- Your network mask ("netmask"). This is a dotted quad, similar to the IP address, which determines which portion of the IP address specifies the subnetwork number, and which portion specifies the host on that subnet. (If you're shaky on these TCP/IP networking terms, we suggest reading some introductory material on network administration.) The network mask is a pattern of bits, which when overlayed onto an address on your network, will tell you which subnet that address lives on. This is very important for routing, and if you find, for example, that you can happily talk to people outside your network, but not to some people within your network, there is a good chance that you have an incorrect mask specified.

  Your network administrators will have chosen the netmask when the network was designed, and therefore they should be able to supply you with the correct mask to use. Most networks are class C subnetworks which use 255.255.255.0 as their netmask. Other Class B networks use 255.255.0.0. The NET-2 code will automatically select a mask that assumes no subnetting as a default if you do not specify one.

  This applies as well to the loopback port. Since the loopback port's address is always 127.0.0.1, the netmask for this port is always 255.0.0.0. You can either specify this explicitly or rely on the default mask.

- Your network address. This is your IP address masked bitwise-ANDed the netmask. For example, if your netmask is 255.255.255.0, and your IP address is 128.253.154.32, your network address is 128.253.154.0. With a netmask of 255.255.0.0, this would be 128.253.0.0.

  If you're only using loopback, you don't have a network address.

- Your broadcast address. The broadcast address is used to broadcast packets to every machine on your subnet. Therefore, if the host number of machines on your subnet is given by the last byte of the IP address (netmask 255.255.255.0), your broadcast address will be your network address ORed with 0.0.0.255.

For example, if your IP address is 128.253.154.32, and your netmask is 255.255.255.0, your broadcast address is 128.253.154.255.

Note that for historical reasons, some networks are setup to use the network address as the broadcast address, if you have any doubt, check with your network administrators. (In many cases, it will suffice to duplicate the network configuration of other machines on your subnet, substituting your own IP address, of course.)

If you're only using loopback, you don't have a broadcast address.

- Your gateway address. This is the address of the machine which is your "gateway" to the outside world (i.e. machines not on your subnet). In many cases the gateway machine has an IP address identical to yours but with a ".1" as its host address; e.g., if your IP address is 128.253.154.32, your gateway might be 128.253.154.1. Your network admins will provide you with the IP address of your gateway.

  In fact, you may have multiple gateways. A *gateway* is simply a machine that lives on two different networks (has IP addresses on different subnets), and routes packets between them. Many networks have a single gateway to "the outside world" (the network directly adjacent to your own), but in some cases you will have multiple gateways—one for each adjacent network.

  If you're only using loopback, you don't have a gateway address. The same is true if your network is isolated from all others.

- Your nameserver address. Most machines on the net have a name server which translates hostnames into IP addresses for them. Your network admins will tell you the address of your name server. You can also run a server on your own machine by running `named`, in which case the nameserver address is 127.0.0.1. Unless you absolutely *must* run your own name server, we suggest using the one provided to you on the network (if any). Configuration of `named` is another issue altogether; our priority at this point is to get you talking to the network. You can deal with name resolution issues later.

  If you're only using loopback, you don't have a nameserver address.

SLIP users: You may or may not require any of the above information, except for a nameserver address. When using SLIP, your IP address is usually determined in one of two ways: Either (a) you have a "static" IP address, which is the same every time you connect to the network, or (b) you have a "dynamic" IP address, which is allocated from a pool available addresses when you connect to the server. In the following section on SLIP configuration this is covered in more detail.

NET-2 supports full routing, multiple routes, subnetworking (at this stage on byte boundaries only), the whole nine yards. The above describes most basic TCP/IP configurations. Yours may be quite different: when in doubt, consult your local network gurus and check out the man pages for `route` and `ifconfig`. Configuring TCP/IP networks is very much beyond the scope of this book; the above should be enough to get most people started.

5.3.2.2 The networking `rc` files

`rc` files are systemwide configuration scripts executed at boot time by `init`, which start up all of the basic system daemons (such as `sendmail`, `cron`, etc.) and configure things such as the network parameters, system hostname, and so on. `rc` files are usually found in the directory `/etc/rc.d` but on other systems may be in `/etc`.

Here, we're going to describe the `rc` files used to configure TCP/IP. There are two of them: `rc.inet1` and `rc.inet2`. `rc.inet1` is used to configure the basic network parameters (such as IP addresses and routing information) and `rc.inet2` fires up the TCP/IP daemons (`telnetd`, `ftpd`, and so forth).

Many systems combine these two files into one, usually called `rc.inet` or `rc.net`. The names given to your `rc` files doesn't matter, as long as they perform the correct functions and are executed at boot time by `init`. To ensure this, you may need to edit `/etc/inittab` and uncomment lines to execute the appropriate `rc` file(s). In the worst case you will have to create the `rc.inet1` and `rc.inet2` files from scratch and add entries for them to `/etc/inittab`.

As we said, `rc.inet1` configures the basic network interface. This includes your IP and network address, and the routing table information for your network. The routing tables are used to route outgoing (and incoming) network datagrams to other machines. On most simple configurations, you have three routes: One for sending packets to your own machine, another for sending packets to other machines on your network, and another for sending packets to machines outside of your network (through the gateway machine). Two programs are used to configure these parameters: `ifconfig` and `route`. Both of these are usually found in `/etc`.

`ifconfig` is used for configuring the network device interface with the parameters that it requires to function, such as the IP address, network mask, broadcast address and the like. `route` is used to create and modify entries in the routing table.

For most configurations, an `rc.inet1` file that looks like the following should work. You will, of course, have to edit this for your own system. Do *not* use the sample IP and network addresses listed here for your own system; they correspond to an actual machine on the Internet.

```
#!/bin/sh
# This is /etc/rc.d/rc.inet1 -- Configure the TCP/IP interfaces

# First, configure the loopback device

HOSTNAME=`hostname`

/etc/ifconfig lo 127.0.0.1      # uses default netmask 255.0.0.0
/etc/route add 127.0.0.1        # a route to point to the loopback device

# Next, configure the ethernet device. If you're only using loopback or
# SLIP, comment out the rest of these lines.

# Edit for your setup.
IPADDR="128.253.154.32"         # REPLACE with YOUR IP address
NETMASK="255.255.255.0"         # REPLACE with YOUR netmask
NETWORK="128.253.154.0"         # REPLACE with YOUR network address
BROADCAST="128.253.154.255"     # REPLACE with YOUR broadcast address, if you
                                # have one. If not, leave blank and edit below.
```

```
GATEWAY="128.253.154.1"         # REPLACE with YOUR gateway address!

/etc/ifconfig eth0 ${IPADDR} netmask ${NETMASK} broadcast ${BROADCAST}

# If you don't have a broadcast address, change the above line to just:
# /etc/ifconfig eth0 ${IPADDR} netmask ${NETMASK}

/etc/route add ${NETWORK}

# The following is only necessary if you have a gateway; that is, your
# network is connected to the outside world.
/etc/route add default gw ${GATEWAY} metric 1

# End of Ethernet Configuration
```

Again, you may have to tweak this file somewhat to get it to work. The above should be sufficient for the majority of simple network configurations, but certainly not all.

`rc.inet2` starts up various servers used by the TCP/IP suite. The most important of these is `inetd`. `inetd` sits in the background and listens to various network ports. When a machine tries to make a connection to a certain port (for example, the incoming `telnet` port), `inetd` forks off a copy of the appropriate daemon for that port (in the case of the `telnet` port, `inetd` starts `in.telnetd`). This is simpler than running many separate, standalone daemons (e.g., individual copies of `telnetd`, `ftpd`, and so forth)—`inetd` starts up the daemons only when they are needed.

`syslogd` is the system logging daemon—it accumulates log messages from various applications and stores them into log files based on the configuration information in `/etc/syslogd.conf`. `routed` is a server used to maintain dynamic routing information. When your system attempts to send packets to another network, it may require additional routing table entries in order to do so. `routed` takes care of manipulating the routing table without the need for user intervention.

Our example `rc.inet2`, below, only starts up the bare minimum of servers. There are many other servers as well—many of which have to do with NFS configuration. When attempting to setup TCP/IP on your system, it's usually best to start with a minimal configuration and add more complex pieces (such as NFS) when you have things working.

Note that in the below file, we assume that all of the network daemons are held in `/etc`. As usual, edit this for your own configuration.

```
#! /bin/sh
# Sample /etc/rc.d/rc.inet2

# Start syslogd
if [ -f /etc/syslogd ]
then
      /etc/syslogd
fi

# Start inetd
```

```
if [ -f /etc/inetd ]
then
      /etc/inetd
fi

# Start routed
if [ -f /etc/routed ]
then
      /etc/routed -q
fi

# Done!
```

Among the various additional servers that you may want to start in `rc.inet2` is `named`. `named` is a name server—it is responsible for translating (local) IP addresses to names, and vice versa. If you don't have a nameserver elsewhere on the network, or want to provide local machine names to other machines in your domain, it may be necessary to run `named`. (For most configurations it is not necessary, however.) `named` configuration is somewhat complex and requires planning; we refer interested readers to a good book on TCP/IP network administration.

5.3.2.3 `/etc/hosts`

`/etc/hosts` contains a list of IP addresses and the hostnames that they correspond to. In general, `/etc/hosts` only contains entries for your local machine, and perhaps other "important" machines (such as your nameserver or gateway). Your local name server will provide address-to-name mappings for other machines on the network, transparently.

For example, if your machine is `loomer.vpizza.com` with the IP address 128.253.154.32, your `/etc/hosts` would look like:

```
127.0.0.1               localhost
128.253.154.32          loomer.vpizza.com loomer
```

If you're only using loopback, the only line in `/etc/hosts` should be for 127.0.0.1, with both `localhost` and your hostname after it.

5.3.2.4 `/etc/networks`

The `/etc/networks` file lists the names and addresses of your own, and other, networks. It is used by the `route` command, and allows you to specify a network by name, should you so desire.

Every network you wish to add a route to using the `route` command (generally called from `rc.inet1`—see above) *must* have an entry in `/etc/networks`.

As an example,

```
default 0.0.0.0 # default route    - mandatory
```

```
loopnet 127.0.0.0 # loopback network - mandatory
mynet 128.253.154.0 # Modify for your own network address
```

5.3.2.5 /etc/host.conf

This file is used to specify how your system will resolve hostnames. It should contain the two lines:

```
order hosts,bind
multi on
```

These lines tell the resolve libraries to first check the `/etc/hosts` file for any names to lookup, and then ask the nameserver (if one is present). The `multi` entry allows you to have multiple IP addresses for a given machine name in `/etc/hosts`.

5.3.2.6 /etc/resolv.conf

This file configures the name resolver, specifying the address of your name server (if any) and your domain name. Your domain name is your fully-qualified hostname (if you're a registered machine on the Internet, for example), with the hostname chopped off. That is, if your full hostname is `loomer.vpizza.com`, your domain name is just `vpizza.com`.

For example, if your machine is `goober.norelco.com`, and has a nameserver at the address 128.253.154.5, your `/etc/resolv.conf` would look like:

```
domain      norelco.com
nameserver  127.253.154.5
```

You can specify more than one nameserver—each must have a `nameserver` line of its own in `resolv.conf`.

5.3.2.7 Setting your hostname

You should set your system hostname with the `hostname` command. This is usually called from `/etc/rc` or `/etc/rc.local`; simply search your system `rc` files to determine where it is invoked. For example, if your (full) hostname is `loomer.vpizza.com`, edit the appropriate `rc` file to execute the command:

```
/bin/hostname loomer.vpizza.com
```

Note that the `hostname` executable may not be found in `/bin` on your system.

5.3.2.8 Trying it out

Once you have all of these files set up, you should be able to reboot your new kernel and attempt to use the network. There are many places where things can go wrong, so it's a good idea to test individual aspects of the network configuration (e.g., it's probably not a good idea to test your network configuration by firing up Mosaic over a network-based X connection).

You can use the `netstat` command to display your routing tables; this is usually the source of the most trouble. The `netstat` man page describes the exact syntax of this command in detail. In order to test network connectivity, we suggest using a client such as `telnet` to connect to machines both on your local subnetwork and external networks. This will help to narrow down the source of the problem. (For example, if you're unable to connect to local machines, but can connect to machines on other networks, more than likely there is a problem with your netmask and routing table configuration). You can also invoke the `route` command directly (as `root`) to play with the entries in your routing table.

You should also test network connectivity by specifying IP addresses directly, instead of hostnames. For example, if you have problems with the command

```
$ telnet shoop.vpizza.com
```

the cause may be incorrect nameserver configuration. Try using the actual IP address of the machine in question; if that works, then you know that your basic network setup is (more than likely) correct, and the problem lies in your specification of the name server address.

Debugging network configurations can be a difficult task, and we can't begin to cover it here. If you are unable to get help from a local guru we strongly suggest reading the *Linux Network Administrators' Guide* from the LDP.

5.3.3 SLIP Configuration

SLIP (Serial Line Internet Protocol) allows you to use TCP/IP over a serial line, be that a phone line, with a dialup modem, or a leased asynchronous line of some sort. Of course, to use SLIP you'll need access to a dial-in SLIP server in your area. Many universities and businesses provide SLIP access for a modest fee.

There are two major SLIP-related programs available—`dip` and `slattach`. Both of these programs are used to initiate a SLIP connection over a serial device. It is *necessary* to use one of these programs in order to enable SLIP—it will not suffice to dial up the SLIP server (with a communications program such as `kermit`) and issue `ifconfig` and `route` commands. This is because `dip` and `slattach` issue a special *ioctl()* system call to seize control of the serial device to be used as a SLIP interface.

`dip` can be used to dial up a SLIP server, do some handshaking to login to the server (exchanging your username and password, for example) and then initate the SLIP connection over the open serial line. `slattach`, on the other hand, does very little other than grab the serial device for use by SLIP. It is useful if you have a permanent line to your SLIP server and no modem dialup or handshaking is necessary to initiate the connection. Most dialup SLIP users should use `dip`, on the other hand.

`dip` can also be used to configure your Linux system as a SLIP server, where other machines can dial into your own and connect to the network through a secondary Ethernet connection on your machine. See the documentation and man pages for `dip` for more information on this procedure.

SLIP is quite unlike Ethernet, in that there are only two machines on the "network"—the SLIP host (that's you) and the SLIP server. For this reason, SLIP is often referred to as a "point-to-point" connection. A generalization of this idea, known as PPP (Point to Point Protocol) has also been implemented for Linux.

When you initiate a connection to a SLIP server, the SLIP server will give you an IP address based on (usually) one of two methods. Some SLIP servers allocate "static" IP addresses—in which case your IP address will be the same every time you connect to the server. However, many SLIP servers allocate IP addresses dynamically—in which case you receive a different IP address each time you connect. In general, the SLIP server will print the values of your IP and gateway addresses when you connect. `dip` is capable of reading these values from the output of the SLIP server login session and using them to configure the SLIP device.

Essentially, configuring a SLIP connection is just like configuring for loopback or ethernet. The main differences are discussed below. Read the previous section on configuring the basic TCP/IP files, and apply the changes described below.

5.3.3.1 Static IP address SLIP connections using `dip`

If you are using a static-allocation SLIP server, you may want to include entries for your IP address and hostname in `/etc/hosts`. Also, configure these files listed in the above section: `rc.inet2`, `host.conf`, and `resolv.conf`.

Also, configure `rc.inet1`, as described above. However, you only want to execute `ifconfig` and `route` commands for the loopback device. If you use `dip` to connect to the SLIP server, it will execute the appropriate `ifconfig` and `route` commands for the SLIP device for you. (If you're using `slattach`, on the other hand, you *will* need to include `ifconfig/route` commands in `rc.inet1` for the SLIP device—see below.)

`dip` *should* configure your routing tables appropriately for the SLIP connection when you connect. In some cases, however, `dip`'s behavior may not be correct for your configuration, and you'll have to run `ifconfig` or `route` commands by hand after connecting to the server with `dip` (this is most easily done from within a shell script that runs `dip` and immediately executes the appropriate configuration commands). Your gateway is, in most cases, the address of the SLIP server. You may know this address before hand, or the gateway address will be printed by the SLIP server when you connect. Your `dip` chat script (described below) can obtain this information from the SLIP server.

`ifconfig` may require use of the `pointopoint` argument, if `dip` doesn't configure the interface correctly. For example, if your SLIP server address is 128.253.154.2, and your IP address is 128.253.154.32, you may need to run the command

```
ifconfig sl0 128.253.154.32 pointopoint 128.253.154.2
```

as `root`, after connecting with `dip`. The man pages for `ifconfig` will come in handy.

Note that SLIP device names used with the `ifconfig` and `route` commands are `sl0`, `sl1` and so on (as opposed to `eth0`, `eth1`, etc. for Ethernet devices).

In Section 5.3.4, below, we explain how to configure `dip` to connect to the SLIP server.

5.3.3.2 Static IP address SLIP connections using `slattach`

If you have a leased line or cable running directly to your SLIP server, then there is no need to use `dip` to initiate a connection. `slattach` can be used to configure the SLIP device instead.

In this case, your `/etc/rc.inet1` file should look something like the following:

```
#!/bin/sh
IPADDR="128.253.154.32"         # Replace with your IP address
REMADDR="128.253.154.2" # Replace with your SLIP server address

# Modify the following for the appropriate serial device for the SLIP
# connection:
slattach -p cslip -s 19200 /dev/ttyS0
/etc/ifconfig sl0 $IPADDR pointopoint $REMADDR up
/etc/route add default gw $REMADDR
```

`slattach` allocates the first unallocated SLIP device (`sl0`, `sl1`, etc.) to the serial line specified.

Note that the first parameter to `slattach` is the SLIP protocol to use. At present the only valid values are `slip` and `cslip`. `slip` is regular SLIP, as you would expect, and `cslip` is SLIP with datagram header compression. In most cases you should use `cslip`; however, if you seem to be having problems with this, try `slip`.

If you have more than one SLIP interface then you will have routing considerations to make. You will have to decide what routes to add, and those decisions can only be made on the basis of the actual layout of your network connections. A book on TCP/IP network configuration, as well as the man pages to `route`, will be of use.

5.3.3.3 Dynamic IP address SLIP connections using `dip`

If your SLIP server allocates an IP address dynamically, then you certainly don't know your address in advance—therefore, you can't include an entry for it in `/etc/hosts`. (You should, however, include an entry for your host with the loopback address, 127.0.0.1.)

Many SLIP servers print your IP address (as well as the server's address) when you connect. For example, one type of SLIP server prints a string such as,

```
Your IP address is 128.253.154.44.
Server address is 128.253.154.2.
```

`dip` can capture these numbers from the output of the server and use them to configure the SLIP device.

See Section 5.3.3.1, above, for information on configuring your various TCP/IP files for use with SLIP. Below, we explain how to configure `dip` to connect to the SLIP server.

5.3.4 Using dip

dip can simplify the process of connecting to a SLIP server, logging in, and configuring the SLIP device. Unless you have a leased line running to your SLIP server, dip is the way to go.

To use dip, you'll need to write a "chat script" which contains a list of commands used to communicate with the SLIP server at login time. These commands can automatically send your username/password to the server, as well as get information on your IP address from the server.

Here is an example dip chat script, for use with a dynamic IP address server. For static servers, you will need to set the variables $local and $remote to the values of your local IP address and server IP address, respectively, at the top of the script. See the dip man page for details.

```
main:
  # Set Maximum Transfer Unit. This is the maximum size of packets
  # transmitted on the SLIP device. Many SLIP servers use either 1500 or
  # 1006; check with your network admins when in doubt.
  get $mtu 1500

  # Make the SLIP route the default route on your system.
  default

  # Set the desired serial port and speed.
  port cua03
  speed 38400

  # Reset the modem and terminal line. If this causes trouble for you,
  # comment it out.
  reset

  # Prepare for dialing. Replace the following with your
  # modem initialization string.
  send ATT&C1&D2\\N3&Q5%M3%C1N1W1L1S48=7\r
  wait OK 2
  if $errlvl != 0 goto error
  # Dial the SLIP server
  dial 2546000
  if $errlvl != 0 goto error
  wait CONNECT 60
  if $errlvl != 0 goto error

  # We are connected.  Login to the system.
login:
  sleep 3
  send \r\n\r\n
  # Wait for the login prompt
  wait login: 10
  if $errlvl != 0 goto error
```

```
  # Send your username
  send USERNAME\n

  # Wait for password prompt
  wait ord: 5
  if $errlvl != 0 goto error

  # Send password.
  send PASSWORD\n

  # Wait for SLIP server ready prompt
  wait annex: 30
  if $errlvl != 0 goto error

  # Send commands to SLIP server to initate connection.
  send slip\n
  wait Annex 30

  # Get the remote IP address from the SLIP server. The `get...remote'
  # command reads text in the form xxx.xxx.xxx.xxx, and assigns it
  # to the variable given as the second argument (here, $remote).
  get $remote remote
  if $errlvl != 0 goto error
  wait Your 30

  # Get local IP address from SLIP server, assign to variable $local.
  get $local remote
  if $errlvl != 0 goto error

  # Fire up the SLIP connection
done:
  print CONNECTED to $remote at $rmtip
  print GATEWAY address $rmtip
  print LOCAL address $local
  mode SLIP
  goto exit
error:
  print SLIP to $remote failed.

exit:
```

`dip` automatically executes `ifconfig` and `route` commands based on the values of the variables `$local` and `$remote`. Here, those variables are assigned using the `get`... `remote` command, which obtains text from the SLIP server and assigns it to the named variable.

If the `ifconfig` and `route` commands that `dip` runs for you don't work, you can either run the correct commands in a shell script after executing `dip`, or modify the source for `dip` itself. Running `dip` with the `-v` option will print debugging information while the connection is being set up, which should help you to determine where things might be going awry.

Now, in order to run `dip` and open the SLIP connection, you can use a command such as:

```
/etc/dip/dip -v /etc/dip/mychat 2>&1
```

Where the various `dip` files, and the chat script (`mychat.dip`), are stored in `/etc/dip`.

The above discussion should be enough to get you well on your way to talking to the network, either via Ethernet or SLIP. Again, we strongly suggest looking into a book on TCP/IP network configuration, especially if your network has any special routing considerations, other than those mentioned here.

5.4 Networking with UUCP

UUCP (UNIX-to-UNIX Copy) is an older mechanism used to transfer information between UNIX systems. Using UUCP, UNIX systems dial each other up (using a modem) and transfer mail messages, news articles, files, and so on. If you don't have TCP/IP or SLIP access, you can use UUCP to communicate with the world. Most of the mail and news software (see Sections 5.5 and 5.6) can be configured to use UUCP to transfer information to other machines. In fact, if there is an Internet site nearby, you can arrange to have Internet mail sent to your Linux machine via UUCP from that site.

The *Linux Network Administrator's Guide* contains complete information on configuring and using UUCP under Linux. Also, the Linux UUCP HOWTO, available via anonymous FTP from `sunsite.unc.edu`, should be of help. Another source of information on UUCP is the book *Managing UUCP and USENET*, by Tim O'Reilly and Grace Todino. See Appendix A for more information.

5.5 Electronic Mail

Like most UNIX systems, Linux provides a number of software packages for using electronic mail. E-mail on your system can either be local (that is, you only mail other users on your system), or networked (that is, you mail, using either TCP/IP or UUCP, users on other machines on a network). E-mail software usually consists of two parts: a *mailer* and a *transport*. The mailer is the user-level software which is used to actually compose and read e-mail messages. Popular mailers include `elm` and `mailx`. The transport is the low-level software which actually takes care of delivering the mail, either locally or remotely. The user never sees the transport software; they only interact with the mailer. However, as the system administrator, it is important to understand the concepts behind the transport software and how to configure it.

The most popular transport software for Linux is `Smail`. This software is easy to configure, and is able to send both local and remote TCP/IP and UUCP e-mail. The more powerful `sendmail` transport is used on most UNIX systems, however, because of its complicated setup mechanism, many Linux systems don't use it.

The Linux Mail HOWTO gives more information on the available mail software for Linux and how to configure it on your system. If you plan to send mail remotely, you'll need to understand either TCP/IP or UUCP, depending on how your machine is networked (see Sections 5.3 and 5.4). The UUCP and TCP/IP documents listed in Appendix A should be of help there.

Most of the Linux mail software can be retrieved via anonymous FTP from `sunsite.unc.edu` in the directory `/pub/Linux/system/Mail`.

5.6 News and USENET

Linux also provides a number of facilities for managing electronic news. You may choose to set up a local news server on your system, which will allow users to post "articles" to various "newsgroups" on the system...a lively form of discussion. However, if you have access to a TCP/IP or UUCP network, then you will be able to participate in USENET—a worldwide network news service.

There are two parts to the news software—the *server* and the *client*. The news server is the software which controls the newsgroups and handles delivering articles to other machines (if you are on a network). The news client, or *newsreader*, is the software which connects to the server to allow users to read and post news.

There are several forms of news servers available for Linux. They all follow the same basic protocols and design. The two primary versions are "C News" and "INN". There are many types of newsreaders, as well, such as `rn` and `tin`. The choice of newsreader is more or less a matter of taste; all newsreaders should work equally well with different versions of the server software. That is, the newsreader is independent of the server software, and vice versa.

If you only want to run news locally (that is, not as part of USENET), then you will need to run a server on your system, as well as install a newsreader for the users. The news server will store the articles in a directory such as `/usr/spool/news`, and the newsreader will be compiled to look in this directory for news articles.

However, if you wish to run news over the network, there are several options open to you. TCP/IP network-based news uses a protocol known as NNTP (Network News Transmission Protocol). NNTP allows a newsreader to read news over the network, on a remote machine. NNTP also allows news servers to send articles to each other over the network—this is the software upon which USENET is based. Most businesses and universities have one or more NNTP servers set up to handle all of the USENET news for that site. Every other machine at the site runs an NNTP-based newsreader to read and post news over the network via the NNTP server. This means that only the NNTP server actually stores the news articles on disk.

Here are some possible scenarios for news configuration.

- You run news locally. That is, you have no network connection, or no desire to run news over the network. In this case, you need to run C News or INN on your machine, and install a newsreader to read the news locally.
- You have access to a TCP/IP network and an NNTP server. If your organization has an NNTP news server set up, you can read and post news from your Linux machine by simply

installing an NNTP-based newsreader. (Most newsreaders available can be configured to run locally or use NNTP). In this case, you do not need to install a news server or store news articles on your system. The newsreader will take care of reading and posting news over the network. Of course, you will need to have TCP/IP configured and have access to the network (see Section 5.3).

- You have access to a TCP/IP network but have no NNTP server. In this case, you can run an NNTP news server on your Linux system. You can install either a local or an NNTP-based newsreader, and the server will store news articles on your system. In addition, you can configure the server to communicate with other NNTP news servers to transfer news articles.

- You want to transfer news using UUCP. If you have UUCP access (see Section 5.4), you can participate in USENET as well. You will need to install a (local) news server and a news reader. In addition, you will need to configure your UUCP software to periodically transfer news articles to another nearby UUCP machine (known as your "news feed"). UUCP does not use NNTP to transfer news; simply, UUCP provides its own mechanism for transferring news articles.

The one downside of most news server and newsreader software is that it must be compiled by hand. Most of the news software does not use configuration files; instead, configuration options are determined at compile time.

Most of the "standard" news software (available via anonymous FTP from `ftp.uu.net` in the directory `/news`) will compile out-of-the box on Linux. Necessary patches can be found on `sunsite.unc.edu` in `/pub/Linux/system/Mail` (which is, incidentally, also where mail software for Linux is found). Other news binaries for Linux may be found in this directory as well.

For more information, refer to the Linux News HOWTO from `sunsite.unc.edu` in `/pub/Linux/docs/HOWTO`. Also, the LDP's *Linux Network Administrator's Guide* contains complete information on configuring news software for Linux. The book *Managing UUCP and Usenet*, by Tim O'Reilly and Grace Todino, is an excellent guide to setting up UUCP and news software. Also of interest is the USENET document "How to become a USENET site," available from `ftp.uu.net`, in the directory `/usenet/news.announce.newusers`.